A-Shell

Command Reference

Copyright © 2005 Jack McGregor. All rights reserved.
This edition was produced on 31 August 2005.

Publisher

MicroSabio
22704 Ventura Blvd., #476
Woodland Hills, CA 91364 USA
Web site: www.microsabio.com
Email: info@microsabio.com
voice: 818-710-8437
fax: 818-704-4882

Authors

Jack McGregor has been developing and documenting A-Shell since 1994, and his authorship continues through the date of this publication. Ty Griffin is responsible for editing, formating and publishing this document in its various forms.

Related Information

This document contains the most up-to-date information that was available at the time of publication. It is part of the overall A-Shell documentation set, which consists of the following major components:

- Set-Up Guide
- Command Reference
- Development Guide
- XCALL Reference
- Update Notes

Other related materials include the *A-Shell Telnet Service Guide*, the *A-Shell COM Interface Reference*, and the *Ashlite Program Reference*. All of these publications, in various formats, are available from the MicroSabio web site documents page. Additional information about A-Shell may be included in files that accompany the distributed (i.e., installation) copies of the software.

The documents in the A-Shell collection are available in multiple formats: HTML, Windows Help (CHM) and PDF, to name the most common. If you do not easily find the document format you prefer, please send a request for that format to MicroSabio.

Legal Notice

This documentation and the software it describes contain proprietary information belonging to MicroSabio, a California proprietorship owned entirely by Jack McGregor. The software and this related information is provided under a license agreement containing restrictions on use and disclosure, and is protected by copyright law. This information is confidential between MicroSabio and the client, and remains the exclusive property of MicroSabio.

Due to continued product development, the information contained herein may change without notice. It is believed to be accurate and reliable, at least at the time of its writing. However, no responsibility for the accuracy, completeness or use of this information is assumed by MicroSabio. If you find any problems in the documentation, please report them to the publisher.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the expressed or licensed permission of MicroSabio.

Trademarks

MicroSabio, A-Shell, INMEMO, INFLD and EZ-SPOOL are trademarks of Jack McGregor.

AIX and Informix are registered trademarks of International Business Machines Corporation.
Alpha Micro, AMOS, and AlphaBASIC are registered trademarks of Alpha Micro Products, Inc.
HP-UX is a trademark of the Hewlett-Packard Company.
Linux is a registered trademark of Linus Torvalds.
Microsoft and Windows are registered trademarks of Microsoft Corporation.
Red Hat is a trademark of Red Hat, Inc.
UNIX is a registered trademark of The Open Group.
ZTERM is a trademark of Rod Hewitt at Cool.stf.

All other products, services, companies, events and publications are trademarks, registered trademarks or service marks of their respective owners.

Contents

Updates 7

Command Processing 9
- Filespecs and Directories 10
- Mapping Filespecs 11
- System Command Syntax 12
- Command Search Path 13
- Binary Compatibility 13
- Wildcard Commands 14
- Native Filespecs 14
- Command Line Editing and Recall 15

LIT Commands 17

CMD and DO Files 107
- Arguments $0 through $9 108
- Comments 109
- Colon (:) Commands 109
- IF / ELSE / ENDIF Statements 110
- GOTO and EXIT Statements 111
- LOOKUP Statement 111
- Special $ Variables 112
- Global DO File Parameters 113
- K0 - K9 Commands 114
- Input Case 115
- Concatenate Parameters 116

Index 117

Updates

Listed below are descriptions of the changes to the documentation and A-Shell that have occurred since the release of A-Shell 4.8 in October 2003.

Build, Date	Topic	Description
938, 25 Aug 05	COLOFFSET	Add ability to rename VUE column 0 to column 1
931, 25 May 05	COMPIL	Add DIMX
924, 29 Mar 05	COMPIL	Add RC switch
923, 15 Mar 05	COMPIL	Add Hex, Octal, ASCII Values and DEFINE to *PLUS* Extensions; add ++INCLUDE Nesting to A-Shell Extensions
914, 23 Jan 05	SET	Add DEBUG switch
914, 23 Jan 05	COMPIL	Add TPRINT, DPRINT, DEBUG to A-Shell Extensions
912, 12 Jan 05	SET	Add ATE and SOCKS to TRACE options.
912, 12 Jan 05	SYSTAT	Add /M to display free memory by job
911, 6 Jan 05	SET	Add INFDEF to display or change INFLD codes
910, 3 Jan 05	CMD/DO Files	Add ability to Concatenate Parameters
910, 3 Jan 05	ATECTL	Add new command for ATE "reverse channel"
909, 19 Dec 04	CMD/DO Files	Implement K0 - K9 Command and Input Case
906, 8 Nov 04	COMPIL	Add WRITECD
906, 8 Nov 04	SYSTAT	Add /I to show IP addresses
906, 8 Nov 04	SET	LONGDIR forces DIR to display long filenames
904, 13 Oct 04	SET	Add TERMINAL options to turn off status bars
902, 27 Sep 04	COMPIL	Add XGETARG, XGETARGS, XPUTARG
902, 27 Sep 04	COMPIL	New switches Band MX
897, 10 Aug 04	APPEND	Handle long filenames containing spaces
897, 10 Aug 04	DIR	Add LONG switch
897, 10 Aug 04	SET	Add TERMINAL options to link colors to Windows
879, 12 Apr 04	VUE	Add several new features
877, 1 Apr 04	SET	Allow more convenient set/unset of AUTOPRINT
870, 3 Mar 04	LICENS	Add /ATESRV switch

Build, Date	Topic	Description
868, 25 Feb 04	SRCCOM	Now includes the list file name and switches
868, 23 Feb 04	FIXTRN	Now supports a /D switch for decimal mode
859, 14 Jan 04	COMPIL	Invokes /A automatically as needed; see Switches
864, 04 Feb 04	DO	Increase max expanded DO file to about 122K
857, 23 Dec 03	COMPIL	Supports new A and AV Switches
854, 21 Nov 03	LOAD	Maximum modules per job now 96 (from 32)
864, 05 Feb 04	FORCE	Support AMOS-style multi-line syntax
859, 14 Jan 04	SET	Add STRICT function
857, 23 Dec 03	VERSYS	Show program header flags for RUN and LIT files
851, 05 Oct 03	MMAP	New command lists files to be memory mapped

Command Processing

Although A-Shell is primarily an application development and runtime environment, it also contains an emulation of the AMOS monitor mode (aka "dot prompt") for the benefit of applications, developers, and users that rely on monitor mode for auxiliary operations. Examples of such operations include copying and erasing files, creating PPNs, performing maintenance on ISAM files, getting system status, building function key translation tables, displaying directory listings, dumping files, and logging to different PPNs—and of course, running, compiling, and editing programs.

The operations just described are all implemented via program files, which under both AMOS and A-Shell, have the extension "LIT" and are thus sometimes referred to as "LIT commands." Under AMOS, the LIT name was thought to derive from the idea that these were "literal" commands (i.e. machine language to be executed literally by the CPU) as opposed to commands that are interpreted by some higher-level software (like BASIC programs). Under A-Shell, they are implemented in BASIC and thus the LIT extension may be a bit misleading, but we decided to stick with it nonetheless since the distinction is no longer relevant.

Some of the A-Shell LIT commands are essentially replacements for existing host operating system utilities (e.g. COPY, DIR, ERASE), but others are so AMOS or A-Shell-oriented that there would be no host operating system equivalent (e.g. ISMDMP, ISMUTL, SUBMIT, SYSTAT, etc). Even for the first category, the many differences between AMOS and Windows or UNIX syntax make the A-Shell emulations of the AMOS LIT commands worthwhile. For example, what would be the Windows/UNIX equivalent of the command

Although LIT commands are useful in themselves and often typed directly by users at the "dot prompt", they are more commonly grouped together in CMD or DO files, which are conceptually equivalent to BAT or script files under Windows or shell scripts under UNIX. Due to the obvious utility of being able to create a complex series of command operations that can be invoked via a single command line, they are extensively used in most existing applications. This, perhaps more than any other reason, is the justification for implementing so many of the AMOS LIT commands in A-Shell—i.e., so that existing CMD and DO files would not have to be rewritten when porting applications from AMOS. A section on the details of writing CMD and DO Files is included at the end of this document.

Filespecs and Directories

AMOS organizes disk space into physical devices, logical units, and a pair of octal numbers referred to as the "PPN". Within this structure are the files, which consist of a one-to-six character name and a one-to-three character extension. The characters used in filenames are from the "RAD50" set, which consists of 0-9, A-Z, and $. No part of the filespec is case sensitive. For example, the following are equivalent:

```
DSK0:COPY.LIT[1,4]
dsk0:copy.lit[1,4]
Dsk0:Copy.Lit[1,4]
```

In these examples, the physical device is DSK, the logical unit is 0, and the PPN is [1,4]. The file name is COPY and the extension is LIT.

Under AMOS, a logical unit was essentially a partition within a physical disk device, more or less equivalent to a drive letter under Windows. Within each logical unit, there were individual directories identified by a pair of octal numbers (each ranging from 0-377) referred to as a "PPN" (Project, Program Number). Individual directories (i.e. a PPN within a logical unit within a physical unit) could also be assigned aliases, called ERSATZ devices, which have the syntax of device names. This is similar to the situation under Windows where a drive letter can be mapped to another directory. These ERSATZ devices are defined in a text file DSK0:ERSATZ.INI[1,4]. You can add your own (each one consisting of up to 6 "RAD50" characters starting with an alphabetic character). In addition, the standard ERSATZ.INI defines ERSATZ device names for a collection of PPNs on DSK0: which are considered the "system accounts" and which are listed in the following table.

Device:[P,PN]	ERSATZ	System Account Purpose
DSK0:[1,2]	OPR:	System operator account. This account has special privileges and is usually password protected. Logging into this account might be considered analogous to logging in as Administrator under Windows or as the superuser under UNIX. Certain commands, for example, will only run, or have greater powers, when executed from this account.
DSK0:[1,4]	SYS:	"The" system account. This is the default location for all of the system commands (those with LIT extensions) plus certain configuration files, such as the language files and ERSATZ.INI.
DSK0:[1,6]	DVR:	Terminal driver and language definition file account. Under A-Shell, terminal drivers are embedded in the executable, so the only files you will normally have in this account of the language definition files (e.g. ENGLSH.LDF).
DSK0:[1,7]	ASHCFG:	A-Shell configuration files, in particular, printer init (PQI) files.
DSK0:[2,2]	CMD:	Command files (i.e. those with CMD or DO extensions). These are the AMOS and A-Shell equivalents of BAT files under DOS or shell scripts under UNIX.
DSK0:[7,0]	LIB:	Library files. Function key translation tables, color configuration files, INI.VUE (VUE initialization file).

Device:[P,PN]	ERSATZ	System Account Purpose
DSK0:[7,1]	HLP:	Help files. A-Shell tends to embed help which is associated with system commands in the commands themselves, or use external PDF (Acrobat) or CHM (Windows Compiled Help) files, leaving this directory only for the help files that are displayed from within the VUE command mode with the HELP command.
DSK0:[7,6]	BAS:	RUN files placed in this directory can be run from any other location.
DSK0:[7,376]		A-Shell sample program directory

Mapping Filespecs

Because the non-AMOS operating systems where A-Shell typically runs all use a multi-level hierarchical directory system, we map the AMOS-style devices and PPNs to hierarchical directories using DEVICE statements in the MIAME.INI file; see the *A-Shell Setup Guide* for detailed information about MIAME.INI. But a couple of simple examples should make the concept relatively clear:

```
DEVICE=DSK0 C:\VM\MIAME\DSK0\     ;(Windows)

DEVICE=DSK0 /myapp/dsk0/          ;(UNIX)
```

The above definitions equate the A-Shell device named DSK0 with the directory C:\VM\MIAME\DSK0\ (Windows) and with /myapp/dsk0/ (UNIX). To these base directories, we would convert the PPN into a six digit number, using three digits for the Project number and three digits for the Programmer number. Thus, DSK0:[1,4] would get mapped to C:\VM\MIAME\DSK0\001004\ in the first case and /myapp/dsk0/001004 in the second. Finally, the filename is appended to the end. So, DSK0:COPY.LIT[1,4] would be mapped to C:\VM\MIAME\DSK0\001004\COPY.LIT in the first case (Windows) and to /myapp/dsk0/001004/copy.lit in the second. Under UNIX, we fold the filespec to lower case to be consistent with typical UNIX conventions. Each AMOS or A-Shell device must be mapped to a native path, as in the examples above, but each definition can be entirely independent and arbitrary. It is also possible to map individual PPNs to arbitrary native directories; see the *A-Shell Setup Guide* for more details.

System Command Syntax

The general syntax of an A-Shell system command line is:

```
<command-name> {switches} {arguments} {switches}
```

or, in the case of many commands that operate on files, and/or which accept wildcard filespecs:

```
<command-name> {<outfile>=} {wildcard list} {switches}
```

<Angular brackets> are used in these syntax examples to indicate a token that *must* be replaced, while {curly brackets} indicate items that *may* be replaced (i.e., are optional). So in the example above, {<outfile>=} indicates an optional term; if specified, <*outfile*> must be replaced with an actual argument, such as MYFILE.TXT.

The command name (e.g. **COPY**) is never case sensitive, and is generally given as just the name, without the extension or the directory location. Specifying more than just the name may alter the search path (which see below). If the command takes arguments, they are usually not case sensitive, but this could be up to the discretion of the command implementation.

Switches are always preceded by a forward slash, and are virtually never case sensitive. In most cases, the switch may be abbreviated to the minimum number of unique characters for that command. For example, many commands support the switch */query*, which can usually be abbreviated to */q*. In some cases switches may precede the arguments, but the most common position is at the end.

The <*outfile*=> syntax element in the second example above indicates that the output of the command will be sent to that file, rather than to the screen. This is roughly the AMOS equivalent of the UNIX technique of redirecting *stdout* to a file.

> For some commands, like COPY and APPEND, <*outfile*=> is mandatory.

> Also note that unlike DOS and UNIX, output file arguments precede the input file arguments—i.e., they are reversed—and are separated by the equal sign.

Command lines are tokenized using both spaces and commas, plus other punctuation characters. For example, the command name itself may be terminated by a space, slash, equal sign, or new line. Arguments for LIT commands are typically separated by commas, while arguments for DO commands must be separated by spaces.

The semicolon marks the beginning of a comment, as in this example:

```
DIR/V MYAPP.LST=A.RUN,B.RUN,C.RUN ;this is a comment
```

This would create the file MYAPP.LST containing the directory information including version (/V) of the three specified files.

Command Search Path

When only the command name (i.e. no device, PPN, or extension) is specified, the search path for the command is given in the table below:

Filespec searched for	Notes
DSK0:<name>.LIT[1,4]	LIT command in the system directory.
<name>.LIT	LIT command in the current PPN
<name>.LIT[p,0]	LIT in the ,0 PPN of the current project
DSK0:<name>.CMD[1,4]	CMD file in the system directory
DSK0:<name>.CMD[2,2]	CMD file in the CMD file account
<name>.CMD	CMD file in the current PPN
<name>.CMD[p,0]	CMD file in the ,0 PPN of the current project
<name>.DO	DO file in the current PPN
<name>.DO[p,0]	DO file in the ,0 PPN of the current project
DSK0:<name>[2,2]	DO file in the CMD file account

If the command extension is included with the command name, then the search path is modified from the above to first look for the command in the current PPN (rather than in the system or CMD file directory).

Binary Compatibility

While A-Shell's LIT commands are binary compatible across A-Shell platforms, they are not binary compatible with the AMOS LIT commands. In fact, as previously mentioned, A-Shell's LIT commands are simply BASIC programs that have been renamed from .RUN to .LIT, and which have appropriate startup logic to process command line arguments. You can create your own LIT commands by doing the same thing.

Wildcard Commands

Many of the A-Shell LIT commands support wildcard arguments. Those that do are marked with an asterisk in the "Class" column of the alphabetical listing of LIT commands below. Commands which support wildcards will also accept a comma-delimited list of filespecs, each of which may contain wildcards. Supported wildcards are:

Wildcard	Meaning
*	Matches any full or partial token in the specified position. For example, *.RUN matches any filename with a RUN extension. Or AB*.DAT[1,*] would match any filename beginning with AB that has a DAT extension and is in the Project 1. When a portion of a filespec is omitted, it may act like the * wildcard. For example, DIR ABC is equivalent to DIR ABC.* (rather than DIR ABC*.*).
?	Matches one character. If multiple ?? are used together, then the combination will match between 1 and the specified number of question mark characters. For example, A?BC.?? Would match any file any file starting with A, followed by any character, followed by BC, and with any one or two character extension. May also be used with the [P,PN], e.g. *.BAS[150,27?]
ALL:	Special wildcard that matches all devices. For example, ALL:*.BAS[7,6] would match all files with extension BAS in [7,6] on any device.
[]	Matches any PPN. For example, ALL:[]. You may also omit one of the two parts of the PPN, such as [,0] (equivalent to [*,0]).

Native Filespecs

By default, filespecs used in system commands are expected to be in AMOS format. Some commands, including all of the wildcard commands, will allow you to specify a native filespec by enclosing it in quotes. For example:

```
.COPY HOSTS.TXT="/etc/hosts" /NOD

.TYPE "/etc/hosts"

.SIZE "c:\vm\miame\bin\ashw32.exe"

.DIR HASH.DIR="/vm/miame/bin/ashell" /H
```

When using quoted native filespecs, you cannot use wildcards.

Command Line Editing and Recall

Given the syntactic complexity and advanced typing skills involved with some of these command lines, you are likely to commit syntax errors or simply tire of retyping commands that you already typed just a few moments previously. At that point you will appreciate the ability to recall previous command lines and re-edit them. To cycle backwards through the last 10 command lines, hit Control-R (typically the same as the Page Up key). To cycle forwards through them, hit Control-T (Page Down). Once you find the previously-typed command line, you can re-execute it just by hitting ENTER. If instead of just re-executing it, as is, you prefer to edit it first, you can do so using the following keystrokes:

Primary	Alternate	Editing Action
Control-A		Move cursor to previous word
Control-C		Abort the current line (start a new one).
Control-D	Delete	Delete the character under the cursor
Control-F	Insert	Insert a space under the cursor
Control-H	Left arrow	Move cursor to the left
Control-L	Right arrow	Move cursor to the right
Control-N		Move cursor to the end of the line
Control-Q		Toggle insert/replace mode
Control-R	Page Up	Cycle through the previously typed 10 command lines.
Control-T	Page Down	Cycle forward through the previously typed 10 command lines (after having used Control-R to go backwards).
Control-U		Move cursor to start of command line
Control-W		Move cursor to next word
Control-Y		Delete line to the right of the cursor

LIT Commands

The table below shows the LIT commands that are included A-Shell, and most of them are described in detail in the following sections. An asterisk in the "Class" column, below, indicates that the command will accept wildcards.

Note that many of the commands will display usage information if invoked with a /? Switch.

Command	Class	Function
ABOUT	A-Shell	Display information about the current license and environment
APPEND	AMOS	Appends file together
ASTAT	Utility	Abbreviated dynamic job display
ATECFG	JB Tech	Configure A-Shell Terminal Emulator (Windows only)
ATECTL	A-Shell	Connect ATE "reverse channel" socket
C, CLS	Utility	Clear screen
CHAT	A-Shell	(UNIX only) Full screen dialog between two users
COMPIL	AMOS	AlphaBASIC 1.4 compiler
COMPLP	A-Shell	AlphaBASIC Plus compiler. See COMPIL.
COPY	AMOS *	Copy files
CREATE	AMOS	Allocate random data file
DATE	AMOS	Display system date
DEL	AMOS	Deletes modules from user memory
DEVTBL	AMOS	List virtual AMOS devices
DIFF	UNIX	(UNIX only) Compare two source files.
DING	AMOS	Ring bell
DIR	AMOS *	Produce directory of files §
DO	AMOS	Force .CMD or .DO file execution
DUMP	AMOS *	Dump file contents in octal or hex
ELSE	AMOS	Conditional flow in command file
ENDIF	AMOS	Conditional flow in command file
ERASE	AMOS *	Erase files
ERSATZ	AMOS	Display defined ERSATZ devices
EXIT	AMOS	Exit command or DO file

Command	Class	Function
EZPRT	MicroSabio	Print file, using EZ-SPOOL configured menu of printers
EZTYP	MicroSabio	Display text file with full paging control
FLIT		Used internally by A-Shell.
FIXTRN	AMOS	Function key translation table editing utility
FORCE	AMOS	Force keystrokes into another job's input
FREE	AMOS	Display amount of free space on specified device
GOTO	AMOS	Jump to label within command file
HEAD	A-Shell *	Display (or copy) first N lines of a text file
HOST	A-Shell *	Return to host machine command level
IF	AMOS	Conditional flow in command file
ISMBLD	AMOS	Build ISAM 1.0 file
ISMDMP	AMOS	Dump ISAM 1.0 file
ISMUTL	AMOS	Dump/build/examine ISAM Plus file
JOBALC	AMOS	Display current job name
JSTAT	A-Shell	Monitor another job
KILL	AMOS	(UNIX only) Terminate another process
LICENS	A-Shell	Update license information
LOAD	AMOS	Load file into memory
LOG	AMOS	Change current login account
LOGOFF	AMOS	Log off
LOKUTL	A-Shell	Display information about file locks in use
LOOKUP	AMOS	Lookup a file and display message
MAKE	AMOS	Create blank sequential file
MAP	AMOS*	Display user memory
MDO	A-Shell	Alternate version of DO
MEMORY	AMOS	Change user partition size
MMAP	A-Shell*	Display or load list of files to be memory mapped
MOVE	A-Shell *	Move a file (UNIX / Linux only)
OCMPIL	AMOS	AlphaBASIC 1.3 compiler; see COMPIL
OPRE	Debug plc	AlphaBASIC 1.3 pre-compiler
ORUN	AMOS	Run AlphaBASIC 1.3 or 1.4 program (same as RUN)
PAGE	Debug plc	Page command output
PPN	AMOS	Display PPNs on a given device
PRE	Debug plc	AlphaBASIC 1.4 pre-compiler
PRINT	AMOS *	Sends files (wildcard spec) to a spooler
PWD	A-Shell	Display current login directory in native O/S format
QUIT	A-Shell	Same as HOST, but terminates PolyShell parent and all siblings
QUTL	Debug plc	Queue/XLOCK/FLOCK display utility

Command	Class	Function
RENAME	AMOS *	Rename files
RUN	AMOS	Run AlphaBASIC 1.3 or 1.4 program
SAVE	AMOS	Save module from memory to disk
SEND	AMOS	Send text message to another user
SET	AMOS	Set miscellaneous system and job parameters
SIZE	AMOS	Display size of specified file, in bytes
SLEEP	AMOS	Sleep specified number of seconds
SLEEPR	AMOS	Sleep until specified time
SORT	AMOS	Sort sequential data file
SRCCOM	AMOS	Compare two source files
SUBMIT	AMOS *	Submit a background task
SYSACT	AMOS	Maintain PPNs and passwords
SYSTAT	AMOS	Display jobs and devices
TAIL	A-Shell *	Display (or copy) last N lines of a text file
TELNET	JB Tech	Launch A-Shell Terminal Emulator; see ATECFG
TELSER	A-Shell	Turn current session into telnet server
TIME	AMOS	Display system time
TYPE	AMOS	Type sequential text file on screen
U	AMOS	Store a command line for later recall
VER	AMOS	Display current A-Shell version
VERIFY	AMOS	Verify files against a DIR file
VERSYS	A-Shell *	Display detailed version information about a LIT command
VUE	AMOS	Full screen-based text editor
WSET	Debug plc	Install TRACKER 2.0
XY	AMOS	Execute TCRT code
ZXPWIN	A-Shell	(ZTERM utility) Transfer file to PC and launch app

ABOUT

ABOUT {/l}

ABOUT displays information about your license. The /L switch causes the output to written to the file ABOUT.LST instead of the screen. The output looks like this:

```
A-Shell/32/Windows/32 Ver. 4.9(922)-3
Serial #                    4681
Licensed to:                MicroSabio In House
Nodes Licensed:             5
Nodes in Use:               1
Jobs in Use:                1
License Options:            D-Isam ATS ATE COM-XCALL W32IO
Release Date:               6/Mar/2005
Maintenance Expiration:     8/Jun/2006
INI File:                   c:\vm\miame\miame.ini
Key:                        OGJD-5302-UYZ-478392-8C78FHX0
JOBTBL.SYS File:
  Spec:                     c:\vm\miame\jobtbl.sys
  Created:                  07-Mar-05 16:26
```

Details of the software about the software license are also available from, and may be changed with, the LICENS command.

APPEND

```
APPEND {/switches} <outfile>=<infile1>,<infile2>{,…}
```

APPEND creates a file consisting of two or more input files concatenated together.

Switch	Meaning
Ignore	Causes nonexistent input files to be ignored. (Otherwise a missing file would cause the operation to abort.)
Z	Strip EOF (^Z) characters from the source files. DOS files, or files transferred to DOS/Windows with some file transfer software, sometimes end with a ^Z used as an end-of-file marker, which can cause all kinds of havoc in programs not prepared to filter these out.

If the outfile already exists, it will be overwritten. You may specify the same file as both the outfile and one of the infiles.

Update Notes: Version 2.1(113), Build 897, 10 Aug 04

APPEND.LIT now handles long native filenames containing spaces. For example, the following is now legal:

```
.APPEND TEST.LST=TEST.LST,"%TEMP%\THIS IS A LONG FILE.LST"
```

ASTAT

```
ASTAT {/switches}
```

ASTAT displays a dynamically updated list of jobs currently running along with the current program name. The list is formatted into two columns to fit more jobs on the screen, making it useful on large systems. You may condense the list further by putting the names of programs that are not of interest in the file BAS:IGNORE.PRG (one per line). Thus, if you want to quickly size up if it is safe to do a shutdown or backup, you may want to list those programs that are safe to interrupt (such as the main menu, application logon, etc.). so that only the jobs which are doing something worthy of attention will be shown.

Switch	Meaning
#	(#=1-9) Sets the sleep interval (time between screen updates) in seconds. The default is 1 second. Note that the shorter the interval, the more load this program will place on the system, particularly on a large system.
C	Causes the console or tty name to be displayed for each job instead of the user name.
S	Sorts the entries by job name.
SU	Sorts the entries by user name.
SP	Sorts the entries by program name.

ASTAT was actually developed and then contributed to the A-Shell community by Carl Staff of Diversified Data Systems. Thanks, Carl!

ATECFG

(Windows only) **ATECFG** is a wrapper for the Windows program, ATECFG.EXE, which is installed along with A-Shell/Windows and which allows you to add or edit "ATE configurations". ATE (A-Shell Terminal Emulator) is a facility that allows you to turn your A-Shell/Windows session into a client connected to another machine, over serial or network (telnet) links. This is a fairly bare-bones terminal emulator, not intended to compete with full-blown standalone emulators such as ZTERM, but which is nonetheless convenient and useful in many situations. It is invoked via TELNET.LIT.

In order to use **TELNET** to make a connection to another machine, you must first define a "configuration", which is a collection of parameters describing the connection. Such configurations are stored in the local Registry. The parameters are reasonably self-explanatory to anyone familiar with telnet and serial connections. See the discussion on TELNET for more information about ATE.

ATECTL

```
.ATECTL /switch
```

ATECTL allows you to connect (switch "C"), disconnect ("D"), or query the status ("S") of the ATE "reverse channel." The "reverse channel" is an optional second socket connection between the ATE client and the server used for the purpose of sending response information from the client to the server. The "reverse channel" generally speeds up communication between the server and client, and minimizes conflicts between keyboard activity and the other control information sent between the two machines.

When connected, all non-keyboard communication from the ATE client to the server is sent via the socket, rather than sharing the stdin data stream with the keyboard. The objective is to eliminate interference between keyboard data (especially in the form of typeahead) and control data (such as result codes from INFLD and GUI operations).

There are two steps to activating the connection. The first is to check the option "Accept Connection From Server" on the Misc tab of the ATE configuration dialog and enter an available port number that is visible by the server to the client (requires ATE build 153 or higher). (A high numbered port, above 20000, is recommended.)

The second step is to execute ATECTL/C from the dot prompt of the server after connecting.

Note that if there is a firewall between the server and client (or you are running a software firewall in the ATE client), you will need to open up the port you choose to connect on.

Also note that use of this connection is optional, and can even be turned on and off during a session. When the connection is not available, ATE will revert to using the keyboard channel for passing control.

CHAT

(UNIX only) **CHAT** allows two users to enter into a dialog in which they can type messages back and forth to each other; this is similar to PTCHAT.LIT under PolyTRACK. While in the dialog, their previous instance of A-Shell is put into the background, and automatically restored to the foreground when the dialog is over.

```
CHAT <target name>
```

Target name may be either the job name or the login user name of the person you want to chat with. (You can use the SYSTAT command to help you determine this information.) The target user will then get a pop-up message on his screen asking him if he wants to engage you in dialog. If not, or if he does not respond within 15 seconds, the dialog does not take place. While this is happening, the originating user will have been returned to the dot prompt and can go on running another program. If the target user does engage, at that point, a similar phenomenon will happen with the originating user. (A pop-up dialog appears on the screen, suspending the previous process.) Once the dialog begins, it continues until either user exits by hitting the Escape key. At that point, the dialog is closed on both screens and both users go back to the process that was previously running.

Note that because of UNIX security rules, it is impossible for one user to send signals to another unless the first user has superuser privileges, or unless they are both sharing the same effective user ID. Refer to the *A-Shell Setup Guide* (Installation) for instructions on achieving this.

COMPIL

The A-Shell compilers (**COMPIL** and **OCMPIL**) are optimized AlphaBASIC compilers which generate the same hash-total .RUN object files as their AMOS equivalents, and support most of the same compile-time switches. They typically compile between two and ten times as fast on similar processor speed Windows and UNIX machines. The only differences are in the support for certain switches and in the reporting of certain compile-time errors. For example the *?Undefined line number or label* message will display just the line number or label which is missing, and not the full offending source line. Also, the A-Shell compilers report the line number which errors occur on, similar to C compilers.

Note that although COMPIL.LIT, OCMPIL.LIT, and COMPLP.LIT all have the same hash code, they cannot be renamed because they retrieve their own name from the command line to determine the mode to work in. However, if you insist on using the name COMPIL but getting the OCMPIL functionality, you can erase SYS:COMPIL.LIT and create a CMD:COMPIL.DO which executes OCMPIL.LIT.

For maximum compatibility of RUN files between platforms, OCMPIL is recommended (as it remains fairly stable under both AMOS and A-Shell). Note, however, that ISAMPLUS is not supported by OCMPIL.

COMPLP is equivalent to **COMPIL** except that it also supports some of the AlphaBASIC Plus extensions (such as DO WHILE / LOOP UNTIL, IF / ELSE / ENDIF, etc). This is equivalent to adding the **/X:1** switch to either **COMPIL** or **OCMPIL**. (You would need to use OCMPIL <prog>/X:1, or COMPLP <prog>/I in order to use ISAM 1.0)

The **++include** command is fully implemented. As its argument, it may take either an AMOS-format file specification or a host machine pathname, enabling AlphaBASIC to be used as a true host machine development language.

The A-Shell linked compiler makes use of available memory specified with the MIAME MEMORY command as its work space. An ?Out of memory error thus implies that this value is insufficient. See the description of the host operating system COMPIL command for notes relating to memory.

Switches

The complete list of support switches is provided below. Note that when executing the compiler from the host operating system command level, the switches must begin with a dash, precede the filename, and be lower case. When being executed from the A-Shell dot prompt, they should begin with a slash and follow the filename.

Note that there several switches supported by A-Shell which are not supported by AMOS: **/AV, /S**, **/DB**, **/V:1, /RC, /X:2,** and **/X:3.**

Switch	Meaning
A	Increase size of offsets used for program addresses from 16 to 24 bits. This fixes "transfer address out of range" compiler errors. Beginning with COMPIL version 1.0(112), COMPIL invokes /A automatically (as needed) even if OPTIONS=STRICT.
AV	Allows the "variable index area" to exceed 64K.
B	Send errors messages to <sourcefile>.lst.
D	Enables use of certain d/BASIC statements. See d/BASIC Extensions.
DEBUG	Activate conditional debug code by compiling the variable DEBUG as 1.
I	Causes ISAM 1.0 statements to be accepted.
M	Cause compiler error for any unmapped variables.
MX	Acts like /M except that it reports every line in which an unmapped variable appears.
N	Turns off the display of compiler statistics.
O	Removes the internal line number references from the compiled object code, making it smaller and faster, but eliminating the capability for error handling routines to display the line number of the error.
RC	Preserves Runtime Compatibility for AMOS even when using certain A-Shell extensions.
S	Turns off the display of ++include file names.
T	Display source code as it is being processed. This sometimes makes it easier to track down errors, although the error messages always include the line numbers. You may want to use the PAGE.LIT utility to manage the speed of the display (e.g. PAGE COMPIL ABC/T).
V:1	Attempts to match duplicate AMOS COMPIL.LIT 1.4(309) and higher
X:1	Enables use of certain BASIC Plus syntax elements. These are clarified in more detail in the section on BASIC Plus extensions below.
X:2	Enables the use of certain A-Shell extensions to AlphaBASIC. See A-Shell Extensions.
X:3	Same as X:2, but creates an SBX file instead of RUN.

AV

Allows the "variable index area" to exceed 64K. Furthermore, like with the /A switch (which is implied by /AV), unless OPTIONS=STRICT, the compiler will automatically restart the compilation using /AV if the

program would exceed the 64K variable index limit. Normal variables require 10 bytes of indexing space each (array variables take more), so this limited programs to something less than 6400 variables. With /AV, the limit becomes 256 times greater. /AV is independent of other compiler switches (except /A), so you don't need to go to BasicPlus mode, but any RUN file thus compiled will not be compatible with AMOS, nor with any version of A-Shell prior to 4.9(857). (You will get the error "RUN file is in an incompatible format" if you try to run such a program under AMOS or an earlier A-Shell.)

The AV switch was added in version 1.0(112), A-Shell Build 857 of 23 December 03. Beginning with COMPIL version 1.0(112), A-Shell Build 859 of 14 Jan 04, COMPIL invokes /AV automatically only if STRICT is not set.

B

Background mode. Sends all sends all error messages to the file <sourcefile>.lst instead of the screen. The format of each message is:

```
<filename>, <line#>, <error message>, <source line>
```

This switch was introduced in COMPIL.LIT 1.0(114), A-Shell build 902 of 27 September 2004.

I

In COMPIL (1.4) and COMPLP (BASIC Plus) modes, causes ISAM 1.0 statements to be accepted (and in the case of ambiguous open statements, for ISAM 1.0 to be assumed). New file type open specifiers ISAM'INDEXED{'EXCLUSIVE} and ISAMP'INDEXED{'EXCLUSIVE} can be use to avoid ambiguity and to allow both ISAM 1.0 and ISAMPLUS to coexist in the same program.

MX

Acts like /M except that it reports every line in which an unmapped variable appears, even if that variable has already been reported. Note that, as with other errors, typically only the first error on a line is reported. (Otherwise you might get dozens of errors on each line due to the cascading effect.)

This switch was introduced in COMPIL.LIT 1.0(114), A-Shell build 902 of 27 September 2004.

RC

Enables the use of several compiler and source code enhancements without braking backwards Runtime Compatibility with AMOS. To be effective, you must use with COMPIL.LIT/OCMPIL.LIT 1.0(117)+ (not COMPLP) and without any of the /X:# switches. The idea is to accommodate people who want to compile under A-Shell and use as many A-Shell features as possible, but still retain the option of copying the RUN files to AMOS for execution there.

The features which /RC adds to COMPIL or OCMPIL are:

- DEFINE

- Hex, Octal, ASCII Values
- IF / ENDIF
- Do While...Until Loop (but not EXIT)
- Use of underline in identifiers
- Max source line length of 3070 instead of 512
- ++PRAGMA (all)
- Comment may follow an & on a continuation line
- ++INCLUDE nested 3 deep

RC was added to A-Shell in Build 924 of 29 March 2005.

S

Turns off the display of ++include file names. (This conflicts with the AlphaBASIC Plus compiler which uses /S for .SBR instead of .XBR subroutine extensions. Under A-Shell, subroutine name extensions are irrelevant, since they are linked into the executable.) Note that since eliminating the ++include messages makes it difficult to determine which include file an error occurs in, the error message format is modified by this switch to include the file name and line number.

V:1

Attempts to match new features and quirks of the AMOS version of COMPIL.LIT 1.4(309) and higher (released with AMOS 2.3A). Unfortunately, although this version of COMPIL shares the same major and minor version number (1.4) with preceding versions, it introduces some new features and generates RUN modules with various differences, causing them to be incompatible with earlier versions of RUN.LIT. This is true under both AMOS and A-Shell. The main reason for using this switch would be if you are using a source code feature that was not previously supported. WARNING: Even with the /V:1 switch, the A-Shell compiler does not support all of the AlphaBASIC 1.4 changes and may not generate the same hash as the AMOS version. Also note that /X:1 and /X:2 both implicitly give you /V:1 features. Features that are activated by /V:1 include:

- LOOKUP returns 0.5 for a zero block file
- READ'ONLY option on OPEN statements
- READ'READ'ONLY statement
- ALLOCATE'INDEXED supports the Device param.

PLUS Extensions

The /X:1 switch (or equivalently, COMPLP) extends the compiler support to include certain statements which were originally implemented by BASIC Plus under AMOS; these are listed below. Refer to the AlphaBASIC PLUS User's Manual published by Alpha Microsystems for more details on these language features.

Note that although A-Shell supports these BASIC Plus language extensions in source code, in no way does it support the .RP (compiled BASIC Plus) file format. Instead, it compiles them as non-standard extensions to the .RUN format, and any such .RUN files will not be compatible with RUN.LIT or RUNP.LIT under AMOS. Thus, if you choose to use these language extensions and want to run them under both A-Shell and AMOS, you will have to recompile the source on both platforms.

Extension	Meaning
Do While...Until Loop	
EXIT	EXIT may be placed inside of a DO/WHILE or FOR/NEXT loop to cause an immediate exit of the loop.
For...Next Loop	

IF / ENDIF	IF <condition> THEN
	statements
	ENDIF

EDIT	String editing function
FILL$(pattern, length)	The FILL$ function copies the given expression, partially or multiple times, up to the specified length.
ODTIM	Day / Time output conversion/formatting
READ'ONLY	File open mode modifier
OPEN for APPEND	Enhancement to OPEN #CH, FSPEC, APPEND
I (Integer) variable type	Integer variable type
AMOS	Execute AMOS commands as subroutines
GETKEY	Input a character
CMDLIN	Retrieve command line
DIVIDE'BY'0	Determine how to handle divide by zero
++PRAGMA	Compiler directives
LOOKUP("fspec") function	Similar to LOOKUP statement, but returns file size as value of expression. Note that a zero block file returns 0.5.
ERRMSG(x) function	Returns the text of the BASIC error message associated with the specified BASIC error number (from the SYS:ERRMSG.xxx file).
STRIP(var) function (or STRIP$(var))	Returns a string identical to var except with all of the trailing spaces removed.
INPUT RAW	Raw input, from file channel or keyboard, into X variables.
CHAIN prog arg1 arg2...	The ability to pass command line arguments in a chain statement is supported in all compiler modes by A-Shell.
ISAM'INDEXED {'EXCLUSIVE}, ISAMP'INDEXED {'EXCLUSIVE}	These new file type specifiers may be used in file OPEN statements to clarify if the file is to be handled as ISAM 1.0 or ISAMPLUS. See /I switch above. (Note that this applies to AlphaBASIC 1.4 as well as BASIC Plus.)
Hex, Octal, ASCII Values	Specifying literal values in an alternate radix
DEFINE	Define a symbolic constant

Do While...Until Loop

```
DO {WHILE or UNTIL condition}
    statements
LOOP {WHILE or UNTIL condition}
```

For example:

```
DO WHILE A > B
    A = A + 1
    B = B * 2
LOOP
```

Another example:

```
DO
    A = A + 1
LOOP UNTIL A = 10
```

For...Next Loop

The syntax of For...Next loops is the same under BASIC Plus, but at runtime, the termination condition is evaluated prior to the first iteration, allowing the loop to have no iterations at all. Under BASIC 1.3 and 1.4, For...Next loops always execute the body of the loop at least once.

EDIT$

EDIT$(expr, edit-code)

The EDIT$ function modifies the specified string expression according to the edit-code, which may contain one or more of the following flags:

Flag	Effect
2	Remove spaces and tabs
4	Remove CR,LF,FF,RUB,ESC,NULL
8	Remove leading spaces and tabs
16	Shrink multiple spaces/tabs to one space
32	Remove trailing spaces and tabs

ODTIM

ODTIM (date, time {,flags})

This function outputs the specified date and/or time according to the formatting flags. If date and/or time is zero, the current date/time is used. The flags are the same as those used by ODTIM.SBR. Refer to the *A-Shell Development Guide* or Alpha Micro documentation for details on the flags.

READ'ONLY

This optional modifier can be placed at the end of a random file open statement to open a file for read only access. Any attempts to write to such a file will generate an error. Aside from locking issues, this mode may be used to dramatically speed up file access over a network for processes that require no locking, such as reports. Also see ASFLAG.SBR in the A-Shell XCALL Reference for a way to achieve this effect without using the BASIC Plus compiler extensions.

OPEN for APPEND

When compiled in BASIC Plus mode, an attempt to open a sequential file for APPEND mode will automatically create the file if it does not already exist. (Causes a file-not-found error under previous versions of BASIC.)

I (Integer) variable type

An integer variable type is supported under BASIC Plus. This format is similar to B (binary) variables except for the following:

- I variables come in only 1, 2 and 4 bytes
- I variables may hold negative values
- I variables use Alpha's 68K byte order (high word – low word, low byte – high byte), instead of the strict low-byte to high-byte format used by B variables.

As far as A-Shell is concerned, there is no advantage to this variable type other than compatibility with programs already using it, except in cases where you need to support negative integers.

AMOS

```
AMOS {/switch} <command>
```

Invokes an "AMOS" command from within BASIC. Similar to XCALL AMOS. The optional /switch may be /T or /S to either force tracing or silencing of all output (same idea as :T and :S in CMD/DO files). If omitted, the default is to display the output of commands but not the commands themselves (like :R).

GETKEY

```
GETKEY(X {,xltname})
```

This function inputs and returns a single character. The optional *xltname* argument may be used to specify a function key translation table to process the character through if it is a function key.

CMDLIN

This system variable is automatically set to contain the remainder of the command line used to start the current program. For example, if the program was started with:

```
RUN FALALA do re mi
```

Then A$ = CMDLIN would set A$ to “DO RE MI” (always folded to upper case).

XCALL LSTLIN, A$ on the other hand, would return “RUN FALALA do re mi” (the complete command line with no folding).

DIVIDE’BY’0

This statement, along with its converse, determine how an attempt to divide by zero is handled from that point forward in the program. In the default case (NO’DIVIDE’BY’0) such an attempt causes an error #10. After DIVIDE’BY’0 has been executed, such an attempt will return a quotient of 0.

++PRAGMA

NO’LINE’NUMBERS (same as /O)
ERROR’IF’NOT’MAPPED (same as /M)
FORCE’24’BIT (same as /A)
FORCE_EXT “ext” (force compiler output to use ext extension instead of .RUN)

See A-Shell Extensions below for additional ++PRAGMAs only supported by A-Shell,

Hex, Octal, ASCII Values

Hex, octal, and ASCII radix notation can be used to specify numeric literal values, i.e.:

```
&h100          ! hex 100 (256 decimal)
&o100          ! octal 100 (64 decimal)
'A'            ! ascii A (65 decimal)
```

These can be used in expressions where you would otherwise be able to use numeric literal values.

This was added to A-Shell in Build 923 of 15 March 2005.

DEFINE

```
DEFINE symbol = value
```

This statement defines the constant <symbol> to be equal to <value>. It was added to A-Shell in Build 923 of 15 March 2005. Some examples:

```
DEFINE TCRT'REVON = 32
```

```
DEFINE XTF'CTRLC = &h01000000
DEFINE MY'NAME$ = "JACK"
```

The names following the DEFINE must conform to the normal rules for variable names (start with an alphabetic character and contain alphabetic, numeric and apostrophe or underline characters plus an optional $ or % suffix).

The definition must either be a literal numeric value (as in the first two examples above), or a quoted string value.

These definitions act like macros, substituting the definition into the source code before each line of the program is compiled. Because of this, compiler error messages will usually display the line after macro expansions (which may look strange at first, but since the error messages also show the line number, it allows you to compare your source code to the result after the macro replacements.)

From a logic standpoint, the result is equivalent to using MAP statements, i.e.:

```
MAP1 TCRT'REVON,B,1,32
MAP1 XTF'CTRLC,B,4,&h01000000
MAP1 MY'NAME$,S,4,"JACK"
```

The main advantages of DEFINE over MAP statements are:

- You cannot assign a value to a DEFINEd symbol; thus, they act like true constants.
- DEFINEd constants take up no space in the RUN file unless they are used. Thus it now becomes practical to create ++include files which define constants for virtually every situation, without worrying about the overhead. (They do create some compiler overhead, but is that really something we need to worry about?)
- Even for constants that are referenced in the program, constants generally save space in the RUN file. A non-subscripted MAP statement with an initial value takes 16 bytes, and then it takes another 2 or 3 bytes to reference it in the code. A constant takes zero bytes to define, and generally only 2 bytes to reference. (Depending on the number of significant binary digits in the constant, it may take 4 or 6 bytes to represent. But in the vast majority of cases, DEFINEd constants are small integers or powers of 2 and can be represented in only 2 bytes. Strings require 2 bytes plus their length to reference as constants, so in that case, the use of constants may actually increase the size of the RUN, assuming they are referenced multiple times.)
- Because of the above, converting MAP statements to DEFINEd constants (assuming they are never assigned more than one value) will result in smaller RUN files, with a very slight bonus in performance, at the cost of a slight increase in compile times. But the main advantage comes in programming clarity when you use meaningful symbolic names for constant values in place of literal number or mapped variables.

Including a file containing an unknown set of DEFINEs is generally safe and should have no side effect on an existing program (unless you deliberately use the defined constants). There are, however, some scenarios to be aware of:

- If a DEFINE precedes a MAP statement for the same identifier, the MAP statement will generate a syntax error, because the MAP statement identifier will be replaced by the value of the DEFINEd symbol before it gets compiled. For example:

```
DEFINE CTLOP'ADD = 1
MAP1 CTLOP'ADD,B,2,1
```

Since the CTLOP'ADD in the MAP statement will be replaced by 1, it will be compiled (and a syntax error generated) as if it were:

```
MAP1 1,B,2,1
```

- If a DEFINE conflicts with an unmapped variable, a similar thing will happen when you try to assign a value to the variable:

```
DEFINE CANADA'VAT = .15

CANADA'VAT = .16    ! assignment to unmapped variable
```

  This will also cause a syntax error, because it will look to the compiler as if it were:

```
.15 = .16
```

  (If the unmapped variable is never referenced, you won't get a syntax error, but then again it won't matter to your program operation either.)

- The one case where you might have a problem is if you reference an unmapped (and unassigned) variable in an IF statement, i.e.:

```
IF CANADA'VAT = .16 THEN ...
```

  This will compile as if it were:

```
IF .15 = .16 THEN ...
```

  That is a perfectly legal statement, and won't be what you expected (although your test of an unmapped/unassigned variable probably won't be what you wanted either, unless you were counting on it being zero). To eliminate this possibility, compile your programs with /M !!!

- If you attempt to DEFINE a symbol that has already been MAPPED, you will get a duplicate label error. So it doesn't really matter whether you DEFINE all your symbols first or do all your MAPs first, or mix them together; any conflict between the two will generate a compilation error.

A-Shell Extensions

The /X:2 switch extends the compiler support to include certain statements which are only available under A-Shell (i.e. are not part of any known compatibility standard); these statements are listed below.

The /X:3 switch (added in COMPIL.LIT 1.0(115), plus the OCMPIL and COMPLP variants, in A-Shell Build 907, of 8 November 2004) is the same as /X:2 but creates a .SBX file instead of .RUN.

Note that /X:3 does not disturb an existing .RUN, so it is now possible to compile the same source both into a .RUN and a .SBX, as follows:

```
.COMPIL MYPROG/X:2    ; create MYPROG.RUN
.COMPIL MYPROG/X:3    ; create MYPROG.SBX
```

However, in order to make this workable, you'll need to modify the XCALL.BSI file to merely set a flag indicating if it is running in SBX mode or not, instead of aborting with an error message if run directly using RUN MYPROG. You would then test that flag to decide whether to pick up and return parameters.

Extension	Meaning
DEBUG	Control variable. If /DEBUG compile switch specified, acts as if it equals 1, otherwise 0. If /DEBUG switch not specified, then it may still be activated at runtime by using SET DEBUG.
DIMX	Enhanced dynamic array allocation
DPRINT	Like PRINT but creates a static text control using the current default GUI (proportional) font and with the sunken attribute. (Note that if the /X:2 switch not specified, TPRINT is compiled as PRINT.) Intended for displaying data fields.
INPUT CSV	Input comma separated values
PAD	Pad a string with spaces
TPRINT	Same as DPRINT but without the sunken effect. Also note that SET AUTOTPRINT causes all PRINT statements to behave at runtime as if they were compiled as TPRINT.
RETURN (<expression>)	Return value from XFUNC
WRITECD	Output comma delimited values
WRITETD	Same as WRITECD, except that it outputs TAB delimiters instead of comma delimiters.
XFUNC	Call an external (SBX) function
XGETARG,XGETARGS	Retrieve XCALL params. See XGETARG, XGETARGS, XPUTARG
XPUTARG	Return XCALL params. See XGETARG, XGETARGS, XPUTARG
++PRAGMA ALIGN2	Align MAP1 variables on 2 byte boundaries (default).
++PRAGMA ALIGN4	Align MAP1 variables on 4 byte boundaries.
++PRAGMA ALIGN8	Align MAP1 variables on 8 byte boundaries
++PRAGMA SBX	Same as /X:3 (create an SBX instead of RUN)
++INCLUDE Nesting	Nesting to 3 levels deep (instead of just 1)
Underlines in variable names	Variable names (and constants) may contain underlines. Although this has no special meaning, by convention it is used primarily with constants.

DIMX

DIMX was added to A-Shell in Build 931 of 22 May 2005.

`DIMX Anyname (expr1,...,exprn) {type {,sizeexpr}}`

Examples:

```
DIM XTDATA(COUNT),S,XTLEN

DIM XTDATA(COUNT1,A*B,C)
```

DIMX is an enhanced version of the venerable DIM statement used to allocate an array at runtime. The enhancements of DIMX over DIM are:

- It accepts optional type and size specifiers rather than being limited to F6 and string with size determined by STRSIZ.
- The element size is evaluated at runtime, whereas with DIM, it is evaluated at compile time.
- At allocates the memory needed out of the system memory pool, rather then from the user partition. Memory used by DIMX is automatically released when the program ends.

Warning: DIMX creates a runtime token that will not be recognized by earlier version of the runtime system. So if you compile a program using DIMX in 4.9.931 or higher, and then run it under a previous version of A-Shell, you'll get an "invalid syntax code" (basic error 34) when you hit the DIMX statement.

INPUT CSV

```
INPUT CSV #CH, var1 {,var2 {,…varn}}}
```

Acts much like INPUT #CH (input from a file) except that it expects CSV (Comma Separated Values) format. Actually, despite the name, CSV files can use TAB delimiters as well as comma delimiters. This is quite handy when inputting CSV files created from spreadsheet or other "standard" utility programs. One other important difference between INPUT CSV #CH and INPUT #CH is that INPUT CSV #CH operates on only one line of the file per call (unlike INPUT #CH which operates on a field by field basis without regard to the line terminators). In other words, if the input line contains more fields than there are variables in the INPUT CSV statement, those extra fields are discarded. Conversely, if there are more variables in the INPUT CSV statement than there are fields in the input line, then the extra variables are set to null. This greatly simplifies importing data from disparate sources.

PAD

```
PAD(var, size) or PAD$(var, size)
```

Returns a string equal to the input string var (which must be string format) padded with trailing spaces out to the length specified by size. In other words, this is the opposite of STRIP(var). If size is less than the current length of var then it functions just like the LEFT(var, size) function.

RETURN (<expression>)

This form of the RETURN statement must be used within an SBX routine to return to the calling program with the specified expression as the value of the function called by XFUNC. If the SBX was called via XCALL rather than XFUNC, then this statement will be treated as equivalent to an END (i.e. end the subroutine and return to the caller). See XFUNC in the *A-Shell Developer Guide* for more information on implementing external functions.

XGETARG, XGETARGS, XPUTARG

These statements may be used (in place of the MIAMEX, MX_XCBDATA calls) to retrieve and return values within an SBX routine:

```
XGETARGS ARG1{,ARG2,...{ARGN}}
XGETARG ARGNO,ARG
XPUTARG ARGNO,ARG
```

XGETARGS retrieves the XCALL arguments into the corresponding variables. (The first argument is returned in ARG1, the second in ARG2, etc.) If more arguments were passed than you specify variables for, the extra arguments are ignored. Conversely, if you specify more ARG variables in the XGETARGS statement then were passed by the caller, the excess ARG variables are ignored. In either case, the operation is non-destructive to the original argument list, so you can later go back and retrieve (or re-retrieve) parameters using XGETARGS, XGETARG, or the original MIAMEX,MX'XCBDATA statements.

XGETARG ARGNO,ARG performs the same function as XGETARGS except that it retrieves only the one argument (whose number is specified in ARGNO).

XPUTARG ARGNO,ARG is the reverse of XGETARG. It returns the value of the argument ARG into the caller's argument number ARGNO.

Note that in the case of XPUTARG, the ARG can be a variable or an expression, whereas in the case of XGETARGS or XGETARG, the arguments must be variables. (You can't retrieve an argument into an expression, but you can use an expression to return an argument.)

Type conversions will be made as appropriate if the variable types do not match the types passed to the xcall (but it behooves you to use “appropriate” data types for the circumstances).

The following examples may clarify the equivalencies between the new statements and the old MIAMEX, MX_XCBDATA calls:

```
XGETARGS P1,P2,P3,P4
```

is equivalent to:

```
xcall MIAMEX,MX'XCBDATA,XCBADR,XCBGET,1,P1
xcall MIAMEX,MX'XCBDATA,XCBADR,XCBGET,2,P2
xcall MIAMEX,MX'XCBDATA,XCBADR,XCBGET,3,P3
xcall MIAMEX,MX'XCBDATA,XCBADR,XCBGET,4,P4
```

and is also equivalent to:

```
XGETARG 1,P1
XGETARG 2,P2
XGETARG 3,P3
XGETARG 4,P4
```

For returning arguments,

```
xcall MIAMEX,MX'XCBDATA,XCBADR,XCBPUT,1,A+B/C
```

is equivalent to:

```
XPUTARG 1,A+B/C
```

The main advantage of the new statements is just efficiency, both for the programmer and for the computer. (This is especially true if you use XGETARGS to retrieve all arguments in one statement.)

Note that these new statements do not require XCBADR or any of the other variables set up by XCALL.BSI. However, you may still want to continue including XCALL.BSI at the start of your SBX modules since it does set the variable XCBCNT to the number of parameters passed, and the XCBSTRUCT() array to the types and sizes, and this information can be useful in determining how to interpret the passed parameters. (XGETARGS and XGETARG do not give you any obvious indication when fewer parameters were passed to the subroutine than the subroutine is expecting, so a well-written subroutine should, at a minimum, check XCBCNT to see how many parameters were passed.)

You do need to compile with the /X:2 (A-Shell extensions) switch (but this is true for all SBX modules anyway).

WRITECD

(Write Comma Delimited) Variation of the PRINT statement which is useful for creating industry-standard comma-delimited files. Syntax and operation is identical to the PRINT statement, except that it performs three additional functions which are useful when more than one variable is being output on a line: 1) It automatically inserts a comma between each variable being output; 2) it automatically quotes any string variable which itself contains a comma; and 3) it strips leading and trailing blanks from any string variable (unless accompanied by a formatting mask).

Example:

```
WRITECD #1,5.2,"Nov 1, 1998","   abc  "
```

Outputs:

```
5.2,"Nov 1,1998",abc
```

The maximum size of an individual field needing to be enclosed in quotes is 1021 bytes.

906.2 04 Nov 2004 Update

WRITECD/WRITETD will convert any embedded quotes (") within a field into apostrophes ('), to try to be more compatible with other CSV readers. It will also now put double quotes around any field containing quotes or apostrophes (in addition to commas and tabs, as before).

Also note, although this is not a change, the procedure for breaking up a single line of CSV output into multiple WRITECD statements:

```
WRITECD #CH, A, B, C,;
WRITECD #CH, D, E, F
```

This will output the 6 variables on one line. Note that that the comma after the C and before the semicolon is required. If you omit it, the D output will immediately follow C with no delimiter. If you omit the semicolon from the first line, then you'll get two lines of output.

XFUNC

XFUNC (Sbxname, arg1,...,argn)

XFUNC$ (Sbxname, arg1,...,argn)

XFUNC executes an external function, returning a value. It is similar to XCALL (in that it loads and calls an external function) except that it is an expression (which returns a value) rather than a statement (which does not). *Sbxname* is the name of an SBX subroutine which implements the function, and *arg1* through *argn* are the arguments passed to the SBX routine. The only special requirement for the SBX routine is that it return a value (numeric for XFUNC and string for XFUNC$) via a special form of the RETURN statement (which see above).

++PRAGMA ALIGN8

Align MAP1 variables on 8 byte boundaries. Aligning on 4 or 8 byte boundaries may be more efficient on modern processors, and can be useful for matching data layouts originating in other languages (such as C). But beware of the effect this can have when using map overlays. For example:

```
MAP1 A1,S,5
MAP1 A2,S,5
MAP1 AX,S,16,@A1
```

Under the normal 2-byte alignment, A2 will start in the 6th position of the AX overlay. But with 4 byte alignment, it would start in the 8 position.

++INCLUDE Nesting

++INCLUDE files may be nested up to three levels deep. The limit is arbitrary, but was chosen based on the belief that beyond three levels, things are probably just going to get too confusing to be practical.

As a visual aid to helping you keep track of the nesting level of lines with errors, they include "+", "++", or "+++" in front of the line number, depending on the nesting level. (Previously, the line number was always preceded by "++" for includes.) Also, the "Copying from" message is now indented to show the nesting.

When an ERSATZ specification is used with ++INCLUDE, it becomes the default for any ++INCLUDEs nested below it. So for example, if the main program contains ++INCLUDE ASHINC:ASHELL.BSI and ASHELL.BSI contains ++INCLUDE ASHELL.DEF, then the default location for ASHELL.DEF will be in ASHINC:. If not found there, or if the ++INCLUDE includes an explicit ersatz or dev:[p,pn] specification, the normal rules apply. This feature allows you to split an existing .BSI into a .BSI and a .DEF, without having to hardcode the location of the .DEF file.

This was added to A-Shell in Build 923 of 15 March 2005.

d/BASIC Extensions

The /D switch extends the compiler support to include certain statements which are otherwise only available under the d/BASIC compiler (offered by d/Soft). Note that these comprise only a subset of the language extensions that d/BASIC supports. They were chosen for inclusion within A-Shell purely on the basis of user demand.

Language Element	Notes
RTRIM(var)	Strips the trailing blanks from the specified string variable
AUTOCAP(var)	"Intelligently" capitalizes the specified string. Note that the A-Shell implementation of this is rather minimal; users in need of more "intelligent" capitalization should refer to XFOLD.SBR in the *A-Shell XCALL Reference*.
.ACCOUNT	This "dot" variable evaluates to the user's current ppn (e.g. "100,22")
.DATE	This numeric "dot" variable returns the date in the same numeric format as the standard AlphaBASIC DATE system variable (i.e. 3 byte separated format).
.DEVICE	This "dot" variable evaluates to the user's current device (e.g. "DSK0:")
.EXPERIENCE	This "dot" variable evaluates to the numeric value 100 (since "experience" level is not one of the AMOS job attributes that is emulated under A-Shell).
.JOBNAME	This "dot" variable evaluates to the jobname.
.MONTH	This "dot" variable evaluates to the 3 character abbreviation for the current month.
.PGMNAME	This "dot" variable evaluates to the current program name.
.PGMVERSION	This "dot" variable evaluates to the current program version (e.g. "1.2A(100)")
.TERMINAL	This "dot" variable evaluates to the current TRMDEF name.
.TIME	The "dot" variable evaluates to the current time, in HHMMSS format
.USERNAME	This "dot" variable evaluates to the user name.
.YYMMDD	This "dot" variable evaluates to the current date in YYMMDD (string) format.

WHEN /WEND	`WHEN <condition>`
	`  statements`
	`WEND`
	Note that "WHEN" is treated as interchangeable with the BASIC Plus "IF" and "WEND" as interchangeable with "ENDIF".

COPY

COPY copies files from one place or name to another.

```
COPY <out-wildspec> = <in-wildspec1> {/delete} {/query}
```

Parameters

out-wildspec
may be a single file, or a wildcard specification.

in-wildspec
may be a single file, a wildcard specification, or a comma-delimited list of file or wildcard specifications.

delete
is needed to overwrite an existing output file.

query
switch causes COPY to prompt for a confirmation before copying each of the input files.

Examples

The following example simply makes a copy of FILE1.NEW called FILE1.OLD, overwriting an existing FILE1.OLD if necessary.

```
.COPY FILE1.OLD=FILE1.NEW/D
```

This example copies the specified file from the SYS: directory to the current directory.

```
.COPY=SYS:ERSATZ.INI
```

This makes copy of all of the files with BAS extensions in all of the PPNs of the current device/logical unit, plus the file DSK1:DIRECT.LST. The copies will all be in the current PPN and will all have SAV extensions.

```
.COPY *.SAV=*.BAS[],DSK1:DIRECT.LST
```

The following demonstrates the ability to create new directories as needed while copying files. It copies all of the files from DSK0:[50,*] to the corresponding PPN in DSK1:. Note that the ability to create new directories during the COPY operation is only available if you are currently logged into [1,2]. Furthermore, if the OPTIONS=STRICT setting is specified in the miame.ini, then you may not create files during a COPY command outside of the current project.

```
.COPY DSK1:[]=DSK0:[50,*]
```

CREATE

```
CREATE <file>, <blocks>
```

CREATE allocates and fills a file with the specified number of 512-byte blocks.

Under AMOS, "random" or "contiguous" files are actually different in structure from the normal "sequential" files, and as the name implies, are allocated in contiguous space. Under Windows and UNIX, there is no equivalent concept, nor is there a need for one. All data files can be accessed using either the "random" access method or the sequential method. In fact, a file allocated with **CREATE** could later be accessed as if it were a sequential file. Since there is no perfect way to recognize whether an existing file was created with **CREATE** (and thus should be treated as a "random" file) or via a sequential output process, A-Shell can only guess based on whether the file size is a perfect multiple of 512 bytes. If so, there is a very good likelihood that it was intended to be a random file. If not, then it must be a sequential file. The only significance of this decision is for the DIR/F operation (which indicates contiguous files with a "C"), and for the BASIC LOOKUP statement, which by default returns a negative number to indicate random or contiguous files. (The MIAME.INI setting OPTIONS=ABSLOOKUP eliminates that distinction, causing all files to be reported as positive numbers.)

The OPTIONS=BRKALC switch in the MIAME.INI affects what the newly allocated file is pre-filled with. When set, the file is filled with brackets (]]]]). Otherwise, the contents will be whatever was previously occupying that disk space.

DATE, TIME

The A-Shell implementations of **DATE** and **TIME** differ from AMOS in that it is only possible to display system date (or time) information and not to change it. An attempt to set the date (or time) will result in the following message:

```
?You must use the host operating system to reset the date (or time)
```

DEL

```
DEL <spec> {/unlock}
```

DEL under A-Shell is nearly equivalent to the version under AMOS except that the A-Shell version supports the *unlock* switch, which is necessary to delete modules which have been loaded with the /Lock switch.

DEVTBL

DEVTBL displays the names of the defined A-Shell devices. This is essentially just a list of the DEVICE= statements from the MIAME.INI file.

You can use the **PPN** command on any of the defined devices to display the available PPNs on that device.

DIFF

```
DIFF {listfile=} oldfile, newfile {switches}
```

(UNIX only) **DIFF** is a front-end to the standard UNIX **diff** utility, with is much faster and more powerful than the SRCCOM.LIT utility. Like the **SRCCOM** utility, the standard version lists the lines that are different between the two specified files. The differences between DIFF and SRCCOM are:

- DIFF ignores differences that are just white space (i.e. a TAB and 3 spaces is treated as no different than a single space).
- DIFF has no switches (although you are welcome to add your own!)
- DIFF uses the UNIX-like space delimiter to separate the two filespec arguments, rather than a comma as comma space to separate the files.
- DIFF displays its output in the EZTYP viewer rather than just outputting to the screen.

Since DIFF is simply a front-end to the UNIX diff, and since source code is provided (in the [7,376] directory), you can easily modify it to take advantage of other features supported in the UNIX diff. See the diff man page for details.

The only switch currently supported is /w, which causes DIFF to ignore differences comprising white space only.

If you create your own variation of DIFF.LIT, it is recommended that you rename it in order to avoid having it be overwritten by a subsequent update.

DING

```
DING <number-of-times>
```

DING rings the bell. This sophisticated technique is mainly used within CMD and DO files to get the operator's attention, perhaps because input is required or an error has occurred.

Under Windows, and most terminal emulators (such as ZTERM), this will play the "System Default" sound. (You can change the WAV file associated with that sound from the Sound applet of your Control Panel, no doubt amazing and delighting your colleagues by setting it to a shrieking woman, theme from Thus Spake Zarathustra, or a gastronomic by-function.)

If you do not have a sound card, or are using a dumb terminal, you will be at the mercy of the default hardware bell sound (and loudness, or lack thereof).

DIR

Documentation update October 2004; see "LONG" switch below.

```
DIR {listspec=} {fspec1{,...fspecN}} {/switches}
```

The **DIR** command implements most of the switches available under AMOS, although there are several differences that should be noted.

- First, keep in mind that files will only be listed if you have file read permission for them. In the case of UNIX, it is also required that that the directory, and all directories above it have the execute bit set. (The execute privilege on a directory controls whether the directory can be traversed.) This applies equally to all wildcard commands, such as **COPY** and **ERASE**, which will effectively not see files for which you have no read permission.
- Second, unlike under AMOS, where you could use DIRSEQ to sort the files in a directory, it is rather non-standard to sort an actual directory under Windows and UNIX. Instead, the **DIR** program itself is usually relied on to sort the *display* rather than sorting the directory itself. Consequently a set of sorting options has been added to the A-Shell version of **DIR** (see switches beginning with "S" in the table below).
- Third, /MIN and /MAX options have been added (for no particular reason other than that somebody requested it). /ATIME and /ADATE (to display last access time and date) were added just because that information is available under UNIX and Windows.
- Fourth, since **DIR** works by actually scanning the directory structures, it will find files that do not fit the AMOS "six dot three" format and thus cannot be located by non-wildcard commands or by BASIC file open statements. **DIR** will truncate files with names longer than "six dot three" but will use some kind of flag character, such as ">" to give you a clue that there may be more there than meets the eye.

  One particularly nasty problem of this type occurs only under UNIX, which has case-sensitive filenames. Since normal files are by convention named in lower case, A-Shell translates AMOS filespecs (which are not case sensitive) to lower case. But some file transfer programs may produce upper case filenames when transferring files from AMOS to UNIX. Such files will appear in a DIR listing (which folds the displayed filenames to upper case to match the way AMOS DIR listings appear). But they will appear to be invisible to non-wildcard commands and BASIC file option operations. To avoid this problem, make sure your file transfer program has an option to force lower case and that the option is activated. To fix the problem, see the **trlcs** utility which is shipped with A-Shell in the **bin** directory.

In the list of switches below, note that like the AMOS version, you can abbreviate any switch to the minimum number of unique characters (e.g. /C for /CONTIGUOUS, etc.).

Also see the PAGE as an alternative to ^S/^Q for limiting the scrolling speed of the display.

Switch	Meaning
ADATE	Displays last access date.
ATIME	Displays last access time.

Switch	Meaning
CDATE	Displays file “c date."
CONTIGUOUS	Displays a “C” to identify “contiguous” files. (Note that under A-Shell, this really just means the file is a multiple of 512 bytes.)
CTIME	Displays file “c time”. Usually used with /CDATE. See notes under CDATE for differences between the “c date” under AMOS/Windows and UNIX.
DATA	Displays complete filespecs. (See VERIFY for notes on a use for this switch.)
FULL	Same as /H/CON/V
KILL	Delete and replace listfile
LONG	Changes the default display layout to 10.3 format instead of 6.3 format.
MAX:n	Display only files with n or fewer blocks
MIN:n	Display only files with n or more blocks
NOFILES	Omit filenames; just output totals
QUERY	Prompt for confirmation on each input file. (Standard switch)
SEXT	Sort by ascending extensions
SNAME	Sort by ascending filenames
SSIZE	Sort by ascending file sizes
SUDATE	Sort by ascending update dates
UDATE	Display file update (last modified) dates
UTIME	Display file update (last modified) times
VERSION	Display file version if available. (Also see VERSYS)
WIDE:n	Display in n columns (default n=4)

CDATE

Displays file “c date.” The “c date” under AMOS and Windows is the creation date, but under UNIX is the date that the file’s directory entry was last changed. This occurs whenever the file ownership or privileges are changed, or when a symbolic link is created, or when the file is moved, renamed, or copied. Because of this, the “c date” can easily be more recent than the update date.

LONG

Changes the default display layout to accommodate 10.3 format instead of 6.3 format. It also changes the default /W value from 4 columns to 3. As before, filenames exceeding the maximum display length will show a “>” character following the filename. Note that the SET command LONGDIR can be used to force the 10.3 format.

This switch was added in DIR.LIT version 3.0(132), A-Shell build 897.0 of 10 August 2004.

DO

```
DO <fspec>
```

The **DO** command forces *fspec* to be treated as a DO file (rather than a LIT or CMD file). This is useful in situations where the *fspec* in question exists in CMD and/or LIT variations as well as DO, since otherwise, the command search path would cause the CMD or LIT to be found first. It is also needed to force a file with an extension other than DO to be interpreted as a DO file (although it is not clear why you want to do that).

DO.LIT is invoked automatically by the command line processor if it determines that the specified file is a DO file, so it is not necessary to explicitly use "DO" except in the situations described above. **DO.LIT** is responsible for preprocessing the DO file to perform argument and macro substitutions.

MDO.LIT is identical to **DO.LIT**, but is referenced internally under different circumstances.

Update Notes: Version 2.1(117), Build 864.3, 04 Feb 04

DO.LIT and **MDO.LIT** increase the maximum expanded do file from about 62K to about 122K. This is about the maximum size for a 430K partition, which is about the minimum partition needed for some other large LIT commands, such as **SUBMIT**. It will now complain explicitly if the partition size is not sufficient.

Update Notes: Version 2.1(122), Build 910.0.7, 03 Jan 05

When DO is used within a DO file to reference another DO file, the command line may contain references variables input during execution of the current DO file, $KBD0 thru $KBD9 (see K0 - K9 Commands), upper and lower case functions (see Input Case)and use the concentration operator $+ (see Concatenate Parameters).

DUMP

```
DUMP <filespec> {,starting-block}
```

The **DUMP** command is an implementation of the AMOS command of the same name, for displaying the contents of files only; direct disk block dumps, UFD dumps, and MFD dumps are *not* supported. The format of the data dump is identical to that produced by AMOS, formatted in either octal or hexadecimal depending on the current job settings (see SET). The current block number displayed under A-Shell is a sequential count of 512 byte blocks starting at one (or the specified block number).

Since there is no difference between random and sequential files under operating systems other than AMOS, and the block linking mechanism is separated from the data itself, all 512 displayed bytes contain actual data; the first two or four will not contain link information as they do for sequential files under AMOS. This may mean that dumping a sequential file under AMOS and A-Shell (which have the same hash-total) will produce differing outputs, being shifted by two or four bytes each block.

Also see the notes on PAGE for how to keep the display from scrolling too fast.

512 byte “blocks” do not necessarily have any physical significance outside of AMOS. Under A-Shell we simply use it as a convenient logical unit.

EL, EP

```
EL <compil switches> <wildspec> {,wildlist}
```

EL and **EP** launch the A-Shell Editor from within A-Shell; **EL** the Lite version, **EP** the Pro version. When executed, **EL/EP** looks for the initialization file EDITOR.INI in [7,0]; if not found, an option to build one will be offered.

Parameters

compil switches
: should be space delimited and preceded by '-' (a hyphen). The default switches are defined in EDITOR.INI.

wildspec
: is a standard AMOS wildcard spec

wildlist
: is an optional list of zero or more *file.ext* specifications, which may contain wildcards but no device or ppn

Examples

```
.el -x:2 -m prog.bas
.el *.bas
.el mysrc:myfile.bas,*.b?i,*.map
.el -cfg          ; (displays/edits EDITOR.INI)
.el -help         ; (displays help)
.el               ; (launches Editor with no files)
```

Comments

EL/EP will look for additional options in:

```
EDITOR.INI
EDITOR.INI[p,0]
LIB:EDITOR.INI
```

If the "Save View State" button was pressed before ending the previous session of the Editor, then the current session will start with the files previously opened, plus any files mentioned on the command line.

Note that **EL.LIT** and **EP.LIT** are copies of the same program; at runtime, the program looks at its own name to determine which version of the Editor to launch.

ELSE

```
IF <expr>
   <statements to execute if true>
ELSE
   <statements to execute if false>
ENDIF
```

ELSE allows for the flow of control within a CMD or DO file based on various conditions that can be tested. It takes no arguments, and its usage should be clear from the example above. See

CMD and DO Files for more information.

ENDIF

ENDIF is needed to terminate a conditional group of a CMD/DO statements which was started with an IF or **ELSE** statement. See the description on ELSE for an example, and refer to the notes on

CMD and DO Files for more information. Also see the discussion of MOVE for an example of **ENDIF** usage.

ERASE

```
ERASE <wildspec1> {/query}
```

ERASE not surprisingly erases files. The */query* switch causes **ERASE** to prompt for a confirmation before erasing each file matching the wildcard specification.

ERSATZ

ERSATZ displays the list of current ersatz device definitions. These definitions are maintained within one or more text files, each of which is referenced via an **ERSATZ**= statement in the miame.ini file. The ersatz devices may be used syntactically just like real device names.

Additions to the LAST specified ersatz definition file (i.e. the last **ERSATZ**= statement in the miame.ini) will be recognized within several seconds. Additions to any other ersatz definition file will not be recognized until you re-launch your A-Shell session.

EXIT

EXIT terminates a CMD or DO file, and is only needed when you would like to terminate it somewhere other than the physical end of the file (usually in conjunction with the a conditional statement involving **IF**). See the notes on MOVE for an example of **EXIT** usage.

EZTYP

EZTYP <filename>

EZTYP displays the specified file, using the EZ-SPOOL file viewer. Although this command was officially renamed to "**EZVUE**" quite some time ago, the original command name of **EZTYP** has been retained in the interests of backwards compatibility and ontological confusion. Refer to the *A-Shell Setup Guide* for full details on the configuration file mechanism, and note that the colors may be affected by settings in the INI.CLR file.

The operation of **EZVUE/EZTYP** is reasonably self-explanatory, particularly after a glance at the help screen (accessed via the **?** key). However, the topics presented below following may help fill in some of the gaps.

EZVUE can also be accessed via **EZTYP.SBR**, or simply by including the TYPE=ON parameter setting within the applicable EZ-SPOOL configuration file. This latter method will result in an option to preview the file being given every time your application calls **SPOOL.SBR**.

The name **EZVUE** is essentially an alias to **EZTYP** and came into being to highlight the VUE-like features added to the A-Shell version of **EZTYP** (command mode, search capability, paging, etc.).

LOOK is the name of a file viewer included with SuperVue and popular under AMOS, and that **EZTYP** is generally speaking a very suitable (and more capable) replacement for it. Applications which contain hard-coded references to LOOK might want to just create a LOOK.DO file that contains:

```
;LOOK.DO -
$R
EZTYP $0
```

Status Line Information

The status line displays the *current* line, total number of lines, current *page*, total number of pages, and a percentage indicating how far the current page is from the start of the file to the end. This would all be very straightforward, except for some performance considerations which complicate the issue. But first, what is meant by the *current* line? Since **EZVUE** was designed to work with extremely large files, it does not store the file in memory. And since it does not allow changes to the file, there is no *cursor*. So there really is no particular *current* line. However, for the purposes of the status line information, the current position is taken to be the end of the last line visible on the screen.

Next, what is a *page*? If the file has hard page breaks, then an **EZVUE** *page* is equal to a printed page. These page breaks will display as an entire line of ^^ characters on the screen. If the file does not have hard page breaks, then an **EZVUE** *page* is simply a screen-full (typically 21 lines, since 3 lines are used up by the command and status prompts).

Even after defining what a line and a page are, we still have the problem of determining just where we are in the file. This may seem strange, since **EZVUE** looks like an editor, and an editor obviously has to know exactly where you are in the file or you would not be able to edit it with the desired results. The problem, as mentioned above, is that since **EZVUE** was designed to view large report files with the maximum performance and minimum memory overhead (rather than be an editor), it does not load the file into memory. Instead, it uses the operating system's seek facility, which is byte oriented, rather than page or line oriented,

to move about the file. For this reason, it is not able (at least initially) to report the exact number of lines and pages in the file. Instead, it makes an estimate, using the average line size for the part of the file that it has had to scan in order to display it. To avoid giving a false impression of precision, it displays a question mark next to those line and page numbers which are only estimates.

Any example may make this clear. At the start, it only reads enough lines to display one screen full. At that point, it knows exactly which line it is on, but it can only estimate the number of lines and pages in the file, so those figures will have question marks next to them. If you then use the **E** command to go directly to the end of the file, it does a direct seek to the end of the file, then backtracks just enough so that it can display the last lines of the file. At that point, it does not even know which line number or page number it is currently on, since it got to the end of the file without reading it sequentially from the beginning. However, if you request to seek directly to page 50, then it will be forced to scan from the beginning of the file, line by line, until it gets to page 50. At that point, it still does not know the total number of lines or pages, but it does know the current line number and page number, and more importantly, it now knows the position of all of the pages 1-50. Thus, if you then ask to go backward 4 pages, it can go more or less directly to that position without having to start over again from the beginning. If you want to compute the precise number of lines and pages in the file, you can do this by requesting to go directly to a page number which is greater than the number of pages in the file (e.g. **=9999**). After executing that command, it will have been forced to scan the entire file, so from that point forward, it will be able to report the precise number of lines and pages in the file, and be able to go (more or less) directly to any position.

Printing

The print command (**P**) allows you to print an excerpt (from-to page) of the file. This much is obvious. What may not be obvious is where it gets the list of printers that you are allowed to select from. Since this was originally developed as part of EZ-SPOOL, it uses the EZ-SPOOL configuration file system for determining the names of the available printers. This system allows different sets of printers and other options to be associated with various situations, such as job name, program, account, etc.

One potential problem with printing excerpts from a file is that the file may have contained special printer control codes (typically at the start of the file) which are lost when only an excerpt of the file is printed. There is no simple and universal solution to this problem, and you will have to use a workaround approach appropriate to your situation. For example, once you realize that the file is 132 columns wide, you can select a printer whose default configuration is set for 132 column printing. (Refer to the *A-Shell Setup Guide* (Printer.ini files) for information on how you can create a logical printer that automatically starts in a particular configuration.) If your application handles printer control codes via the PREFIX and SUFFIX options in the Printer INI, rather than by embedding them in the file itself, this problem will be minimized. (All you need is enough sense to select the appropriate printer to handle the excerpt.) If you are using a terminal emulator such as ZTERM which offers a choice of printer font sizes for "local" printing, that may be another solution. (In that case, make sure to add a "Local" printer option which uses the "AUXLOC:" pseudo printer device, which in turn will route the printout back to your terminal or emulator for local printing.)

As of A-Shell version 4.7(808), another possibility for dealing with the printing difficulty just described is to use the SBR=EZPRTX parameter in the miame.ini. In this case, whenever you print from within **EZVUE**, it will call the subroutine **EZPRTX.SBX**, passing it the desired page range parameters, and let it handle the print operation. The source to a generic **EZPRTX.SBX** is included in the release in DSK0:[7,376], with the idea that application developers can customize it to provide more sophisticated handling of partial file print requests. (For example, you can add code to deal with the printer feature codes, printer selection, etc.)

Wide reports

EZVUE will automatically switch into 132 column mode if a cursory examination of the first several lines of the file indicates the presence of lines longer than 80 columns (unless AUTOWIDTH=OFF in the EZ-SPOOL configuration file). You can also switch between 80/132 columns with the left and right arrow keys. But if 132 columns is not enough, you can continue to hit the right arrow key to shift the entire display window over, several columns at a time. Hitting the left arrow will reverse this process, finally returning you to 80 column mode when you hit the left margin.

Variable rows

If your terminal supports this capability, Shift-Up Arrow and Shift-Down Arrow (or just T for Tall/Toggle) will toggle you between the normal 24-row display and a 42-row display. On a modern high-resolution display, viewing a large file at 42 x 132 is quite reasonable and much more efficient than at 24 x 80.

FIXTRN

Documentation update July 2004; see "Update Notes" below

FIXTRN {ifx | vux | <name.pfk>}

The **FIXTRN** command performs the functions of the AMOS FIXTRN and SET PFK commands, allowing you to create and edit both types of function key translation tables. The "traditional" types of function key translation table (such as those used by VUE and INFLD) have names that match the terminal driver, with the extension indicating the application. Thus AM62A.IFX would by the translation table for AM62A drivers when running **INFLD**. PCTDV.VUX would be for PCTDV terminal drivers when running VUE. These files are compatible between AMOS and A-Shell. The main difference is that under A-Shell, they are always loaded automatically from the LIB: account on disk, whereas under AMOS, you may need to load them explicitly into system or user memory. To create or edit a function key translation table for **VUE** or **INFLD**, use **FIXTRN** followed by "IFX" for the **INFLD** translation table or "VUX" for the VUE translation table, e.g.:

```
.FIXTRN IFX
   Editing function key translation file: LIB:PCTDV.IFX
```

It will automatically invoke the file for the current terminal driver (PCTDV in the example above) and automatically reference the LIB: directory.

To create or edit a PFK-style function key table (one that can be used in any application and which can be manually loaded and deleted from user memory to override any other translations), specify a filename with a .PFK extension, e.g.:

```
.FIXTRN PILAR.PFK
   Editing function key translation file: PILAR.PFK
```

In this case, it will not put the file in the LIB: directory, since PFK files must be loaded manually, and thus can exist anywhere.

The procedure for editing either type of file is the same (since internally, they are the same format). You start by specifying a special key to be used as the delimiter. Pick one that is not needed in any of your translations, perhaps \. For example:

```
Enter special delimiter key (cannot be used in a translation):\

Hit a function key (or special delimiter to exit): [181]
   Current Translation: HELLO^M
   Update? [y/N] Y
   Enter new translation: LOG GL:^MRUN GLMENU^M
```

In the above example, we hit the F1 key. The program then echoes the numeric value returned by the terminal driver for that key ([181]) and displays the current translation string, if there is one. In the above example, the previous translation was "HELLO" followed by the ENTER key (^M). It then gives you the option of updating the translation. In the above example, we changed the translation to LOG GL: <ENTER> followed by RUN GLMENU <ENTER>. It then loops back to ask for a new key. When you are done examining or defining translations, hit the delimiter key one more time, which will cause it to save the file in the current directory:

```
Hit a function key (or special delimiter to exit):
   Outputting updated PILAR.PFK...
   537 bytes output.
```

To use the translations, use **LOAD** to load the file into memory:

```
.LOAD PILAR.PFK
```

Having done this, you can now hit the F1 key and it will act as if you typed the two commands LOG GL: and RUN GLMENU.

You can use the **MAP** command to check if there is a PFK file currently loaded, and **DEL** to remove it from memory:

```
.MAP
Loaded private function key table: PILAR.PFK
Free       694863

.DEL PILAR.PFK
PILAR.PFK deleted from memory
IFX and VUX modules reset
```

For more information on function keys and translation tables, please refer to *Using A-Shell ... Function Key Translation* in the *A-Shell Setup Guide*.

Update Notes: Version 1.0(102), Build 868, 23 Feb 2004

FIXTRN now supports a /D switch for decimal mode, facilitating the entry and display of binary translations. See **FIXTRN /?** for on-line help.

FORCE

Documentation update June 2004; see "Update Notes" below.

```
FORCE <jobname> <text>
```

FORCE allows you to force keystrokes into another job's input stream. <*text*> is any string of characters. To force control characters, precede them with ^. For example:

```
FORCE TSKAAA RUN PROG^M
```

In the UNIX environment, **FORCE** uses the SIGUSR1 signal, which can only be delivered between processes belonging to the same effective user or if the sender is the superuser. Refer to the *A-Shell Setup Guide* (Installation) for further details on dealing with the UNIX privilege issues. In the Windows environment, **FORCE** uses the IJC (Inter Job Communication) system, which has no privilege issues, but is subject to being turned off by the IJCFREQ command in the MIAME.INI.

Update Notes: Version 2.0A(105), Build 864.4, 05 Feb 04

FORCE.LIT supports the AMOS-style multi-line syntax:, i.e.

```
.FORCE TSKAAB
LOG BAS:
RUN MYPROG
15
~
TIME
```

The above would force TSKAAB to execute "LOG BAS:", then "RUN MYPROG", then "15", and finally a blank line (Tilde by itself is treated as a blank line.) The blank line terminates the force operation. (The **TIME** command would then be executed back in the current job.)

Note that under A-Shell/Windows, the response time for receiving forced commands may be several seconds. To avoid overrunning the target job, the sender will be forced to wait for each command to be acknowledged by the receiver.

It also supports forcing commands to yourself. This is useful for forcing input to a program that does not normally accept input from a command file. It is usually used with the $NJ system variable in DO files to specify the current jobname, e.g.:

```
;MYPROG.DO
:R
LOG BAS:
FORCE $NJ
RUN MYPROG
15
~
TIME
```

Note that the **TIME** command here will execute *before* the forced commands, since CMD and DO input is processed first, then typeahead. Also note that there isn't much point in this exercise unless MYPROG (in this

case) doesn't accept command file input. Otherwise we could just omit the **FORCE** command and MYPROG would run the same from the DO file as it would with the forced input.

A-Shell still supports the non-AMOS standard single-line **FORCE**, in which the command is passed on the command line, e.g.:

```
.FORCE TSKAAB LOG SYS:^M
```

Note that in this mode, you have to explicitly add a ^M to the end of the line to force a carriage return.

Finally, note that the LITMSG.xxx message files have been updated for some new messages in **FORCE**.

HEAD

HEAD {outfile=}<wildspec>{switches}

The **HEAD** command is similar to the eponymous UNIX command and is used to display or copy to another file the first N lines of a specified file (or wildcard list of files). The supported switches are:

Switches

delete
Required when an output file is specified that already exists.

lines:##
Specify the number of lines to copy or display. (Default is 20).

query
Prompt for confirmation on each file.

As with virtually all other A-Shell and AMOS LIT commands, the switches can be abbreviated to the minimum number of unique letters (/L:##, /D, and /Q in the cases above). Also note that virtually all commands which support wildcards support the /Q switch.

Examples

This example displays the first 20 lines of the specified file (more or less equivalent to using PAGE TYPE <file> to display one screen-full and then aborting.

```
.HEAD SYS:ERRMSG.USA
```

This next example is similar, except it uses quotes to refer to a native host operating system file specification (note that they are case sensitive under UNIX) and also uses the /L:5 switch to limit the display to the first 5 lines:

```
.head "/vm/miame/miame.ini" /L:5
```

This next example creates a file called VERDIR.TXT, which consists of the first line of each BASIC source file in the current project. Assuming you adopted a standard of putting something meaningful and consistent on the first line of each source file, this could be a very useful way to compare two sets of source files (i.e. use the command below on each set of files, then sort the resulting header files and compare them using **SRCCOM**).

```
.head verdir.txt=*.bas[,*]/l:1/d
```

See the notes on TAIL for the same idea only counting from the end of the file.

HOST

The **HOST** command is a special A-Shell command program. It exits the A-Shell command shell and returns to either the host operating system command level or host batch or shell-script file which was used to invoke A-Shell.

If an AMOS command sequence is piped into A-Shell (that is if the standard input file has been redirected), for example with the following UNIX command:

```
$ echo "time" | ashell
```

then the command sequence will be executed automatically, and once the end of the standard input file is reached, control will return automatically to the host command level. In this case, at no point does A-Shell produce a command prompt, though it is possible to use the **HOST** command from within a **.CMD** or **.DO** file to terminate the execution of the command sequence prematurely.

The **HOST** command may also be used to perform a host machine function from within A-Shell without exiting, by specifying the command as a parameter. For example:

```
.host vi $0
```

may be used within a **.DO** file on a UNIX machine to invoke the UNIX editor vi. (You will then understand why A-Shell contains an implementation of VUE...)

> TRACKER (which controls the screen output within A-Shell) is not able to monitor any output that occurs outside of A-Shell and thus it is a good idea to resynchronize the display after using a HOST command (by clearing the screen, for example).

Yet another use for the **HOST** command is to execute a host operating system function and redirect the output to a file, which is available for access from within A-Shell. For example, the following commands will create a detailed process status list in the file ps.lst, which is then displayed using the A-Shell **EZTYP.LIT** utility. This technique avoids the terminal handling problems (alluded to above) which are otherwise bothersome when trying to shell out to host operating system functions which do not know anything about the A-Shell display environment.

```
.host ps -ef > ps.lst
.eztyp ps.lst
```

IF

```
IF <expr>
   <statements to execute if true>
ELSE
   <statements to execute if false>
ENDIF
```

IF allows for the flow of control within a CMD or DO file based on various conditions that can be tested. Refer to the

CMD and DO Files for more information.

Example

Here is a simple example of a DO file that takes a filename argument. If the file (with a BAS extension) does not exist, it is created by making a copy of a file called APPLIB:STDHDR.TXT. Then, in either case, the file is VUE'd.

```
IF LOOKUP("$0.BAS") = 0
   COPY $0.BAS=APPLIB:STDHDR.TXT
ENDIF
VUE $0.BAS
```

ISMUTL

ISMUTL <fspec>

ISMUTL allows you to create a new ISAMPLUS file set or perform various maintenance operations on an existing one. ISAMPLUS file sets consist of a data file with a DAT extension, and an index file (containing from one to sixteen logical indexes) with an IDX extension. Since the extensions are fixed, the <*fspec*> should not include an extension, and typically is just a name (e.g. ISMUTL MYFILE).

If the specified file does not exist, you will be prompted for information needed to create the file, including:

```
Number of records to allocate:
Size of data record:
Type of key (0=*, 1=integer, 2=concatenated, 3=binary):
Does key allow duplicates?
Is key allowed to change?
Size of key:
Position of key:
Extra index block percentage:
Loading factor:
Data file device:
```

(The questions relating to keys will be repeated up to 16 times if necessary to define up to 16 indexes.) These prompts are designed to be CMD file compatible with the AMOS version of **ISMUTL**, so that any command files you have to build ISAMPLUS files will work under A-Shell as they do under AMOS. However, the interpretation or significance of some of the parameters differs, as described below.

The **number of records to allocate** is completely ignored under A-Shell, because ISAMPLUS files under A-Shell are automatically self-expanding. The DAT file always starts at 0 bytes, and grows with each record added.

The **size of the data record** is the same as for AMOS, except that the current maximum record size under A-Shell is 8196 bytes.

The "**key allowed to change**" option is presented for CMD file compatibility, but under A-Shell, the key can always change. (It is considered the responsibility of the application to determine this.)

The **extra index block percentage** and **loading factor** options are meaningless under A-Shell, since the index, like the data file, is self-expanding and self-adjusting. (That is not to say that it cannot benefit from reorganization from time to time, but only due to deletions, not to additions alone.)

The **data file device**, is also ignored, as there is no conceivable excuse for putting the DAT file on one device and the IDX on another. (This was an option under AMOS where devices were often no more than 32 MB, but in the Windows and UNIX world, this is hardly a consideration.)

When the specified file set already exists, **ISMUTL** instead displays a menu of maintenance options:

```
LOAD   - load a file
CHANGE - change data device name
ADD2ND - add secondary key directory
DELETE - delete existing secondary key directory
REBUILD- rebuild primary & secondary key indexes
```

```
DUMP   - dump ISAM file
STAT   - display statistical information
ADDIDX - add index block
DESCRP - change key description
EXIT   - exit to AMOS
```

Again, these options are designed to be CMD file compatible with the AMOS version, so that any existing CMD files which execute **ISMUTL** commands will function the same way under A-Shell.

ISAMPLUS under A-Shell is implemented via a clone of the Informix C-ISAM access method, which has many advantages over a homegrown implementation, but does suffer from a few small discrepancies with the AMOS implementation. Some of these were mentioned above. Another is that the data file extension is DAT (rather than IDA as for AMOS). This might create a conflict in programs that explicitly looked for that IDA file (although the MAPIDA option in the miame.ini file may handle most of those). There is also a 1 block LCK file which is used for locking but which does not exist under AMOS. Another difference is that the actual record size on disk will be one byte larger than specified, due to the addition of a control byte. This, and the fact that the file grows on a record-by-record basis (unlike the AMOS version which is preallocated in blocks) makes even the data files (as well as the IDX files) not binary compatible between AMOS and A-Shell. To move them between platforms, you have to either use ISMUTL to dump on the source platform and then **ISMUTL** again to reload from the dump file on the target platform, or you can create a program in BASIC which reconstructs the file from the existing IDA or DAT file (directly, albeit with some programming difficulty).

JSTAT

JSTAT <jobname>

JSTAT is useful for monitoring the status and activity of another job. It displays a dynamically updated screen full of information such as the number of reads, writes and basic instructions, both total and rate/sec over the last sampling interval. It also shows the last basic instruction, last xcall, last line number, current PPN, program, job status flags, etc. This can be useful in a variety of circumstances, such as debugging a job that appears to be in an infinite loop, or determining whether operator is actively keypunching or a remote job is safe to kill.

As with other utilities that rely on inter-job communication, such as KILL, FORCE and SEND, there are privilege issues to consider under UNIX/Linux, whereas under Windows the main issue is the IJCFREQ setting in the MIAME.INI. Refer to the *A-Shell Setup Guide* (UNIX installation) for more information on the privilege issue.

KILL

```
KILL <jobname> {/k}{/y}
```

KILL is similar to, but more complex than, its AMOS counterpart, and works differently depending on whether you are running under Windows or UNIX. The additional complexity arises from the dual nature of jobs running under A-Shell. (Such jobs have both an A-Shell/AMOS nature and an underlying Windows or UNIX process nature.) It also arises from the fact that, unlike the AMOS counterpart, this version of KILL actually tries to clean up resources left in limbo by the target process.

For UNIX, **KILL** starts by checking if the underlying UNIX process still exists. (Under extreme circumstances, it may be possible to abort a UNIX process without A-Shell realizing it; thus leaving a phantom job in the A-Shell job table.) If the UNIX process no longer exists, it offers to clean up the resources (file locks and other queue blocks) still associated with the job. Otherwise, it sends a trappable kill signal (SIGINT, same as Control-C sends) to the target process and waits a few seconds to see if the job goes away. If so, then we are done. If the process does not go away (possibly because the of insufficient privileges) then this fact is reported and nothing further can be done, unless the **/K** switch was specified. With **/K**, and provided that the user is logged into the OPR: account, it will then proceed to use a stronger (but still trappable) kill signal (SIGTERM) to abort the underlying UNIX process (causing a BASIC error 251 in the aborted program). Once this is complete, it checks to see if the job left any resources locked in the qflock.sys or jobtbl.sys files. If so, it prompts you to confirm whether you want them cleaned up.

The **/Y** switch will assume that the answer to this question is "Y", allowing it to be used in unattended processes. An example of where you might want to use this is in a nightly processing routine that requires all users to be either logged out or at least at a known menu. This could be accomplished (under UNIX) by executing **SYSTAT.LIT/L**, and then reading the resulting SYSTAT.LST file and checking to see which jobs are not in the proper state. Then a command file using **KILL <job>** for each such job can be generated and executed. Without the **/K/Y** arguments to **KILL**, it would only have the effect of forcing a Control-C to these jobs. Since this might not be sufficient to force the wayward jobs out of their programs, you could use **KILL <job> /K /Y**. Or if you want to really be fancy, you could try it first without the /K /Y, then run another SYSTAT to see if it succeeded, and only then use **KILL /K /Y** on the stragglers.

> Because of UNIX security rules, it is impossible for one user to send signals to another unless the first user has superuser privileges, or unless they are both sharing the same effective user ID. Refer to the *A-Shell Setup Guide* (Installation) for instructions on achieving this, or on setting the SETUID bit on ashkill so that it runs with root privileges.

Under Windows, **KILL** uses the A-Shell Inter Job Communication (IJC) system to send a message to the target job, effectively telling it, in as friendly a manner as possible, to go kill itself. As with the UNIX version described above, the /K switch determines how lethal the operation is. Without /K, it is basically no different than a Control-C (which may be trapped by the program, or ignored entirely). With /K, the instructions are to abort anything that is running, even VUE. (It is nice enough to save the file first though.) If a BASIC program is running at the time, it will be aborted with error 251 (which can be trapped, but only temporarily, because as soon as the program goes into a keyboard operation it will terminated without any further delay). Note that the IJC system clock frequency is established by the IJCFREQ parameter in MIAME.INI.

LICENS

```
LICENS {/ATESRV}
```

LICENS provides the means to display your A-Shell license information, and to optionally update that information. Other methods of updating the license are to edit the MIAME.INI file directly, and to respond to the prompt which appears on startup if your license has expired.

When executed, **LICENS** displays information about the current license (similar to the ABOUT command), and asks if you want to update the license. It then prompts you for a new serial number, license key, and company name, verifies that they are correct, and asks if you want to save the updated information.

To display the license information, you may be logged into any account. To update the license, you must be logged into the OPR: account.

The **/ATESRV** switch causes **LICENS** to request an ATE license to be issued from the server. This is only applicable when ATE is not licensed on the individual PC workstations. You do not need to be logged in as OPR: when using the **/ATESRV** switch.

Note

If you came to this page wanting to update your license, only to find out that **LICENS** does not run, the problem is that your version of A-Shell predates **LICENS** (version 4.8, build 830, of 30 April 2003). To update the license, you will need to use the "Old Method" described in the *A-Shell Setup Guide*. In brief, you will have to run A-Shell from the host prompt with the start-up switch **-k** (e.g. **ashell -k**), which will prompt you for license information.

LOAD

Documentation update June 2004; see "Update Notes" below.

```
LOAD <fspec> {/lock} {/unlock}

LOAD <common-module-name> /c
```

Prior to A-Shell 4.7, **LOAD.LIT** was mostly a dummy utility used solely to load PFK modules or to clear the COMMON packet. As of 4.7, however, A-Shell supports a "user memory" concept similar to that used in AMOS, and thus the **LOAD** command now works similarly to AMOS as well, except for the special switches.

Switches

lock
: switch "locks" the module in memory so that it cannot be deleted, except by specifying the */unlock* switch on the **DEL** command.

unlock
: is needed in order to reload (update in memory) a module that was previously locked with the */lock* switch.

c
: switch indicates that the specified module name is not a disk file to be loaded into user memory, but the name of a COMMON module, which is to be initialized. This distinction is necessary since COMMON modules continue to be implemented via internal memory buffers within A-Shell rather than via user memory.

Update Notes: Build 854, 21 Nov 03

The maximum number of modules which can be loaded into user memory per job has been increased from 32 to 96.

LOG

LOG {/n} <destination>

A-Shell maps AMOS format file specifications onto host pathnames using the definitions given with the MIAME **DEVICE** commands in the environment configuration file. On entering A-Shell, your current working directory is translated into your current AMOS login account. If there is no mapping defined, then you enter A-Shell in a logged off state. You may use the **LOG** command to log in and subsequently change your login account to one with a valid mapping to an existing host machine directory.

The implementation of **LOG** will check for the validity of the PPN, execute any **START.CMD** command file (unless /N specified), and accept and display ERSATZ names. It does not accept or ask for user names. The user name as returned by the **GETJTB.SBR** subroutine is your user name in the host operating system environment.

The */n* switch disables the normal execution of a START.CMD file in the target directory. *Destination* may consist of all or part of a device and PPN, or an ERSATZ device.

Examples

```
LOG SYS:
LOG DSK0:[1,4]
LOG DSK0:1,4
LOG DSK0:
LOG 1,4
LOG ,4
LOG MYAPP:
```

LOKUTL

When LOKSER-type file locking is on (see the *A-Shell Setup Guide*, Configuration), A-Shell translates LOKSER service requests into the corresponding system locking function supported by the host operating system. Under Windows (non-server versions), there is no convenient function to display a list of the current file locks. However, under some flavors of UNIX (including AIX and Linux), a shareware utility, *lslk*, is available which scans the system locking tables and produces a list of locks. For those flavors of UNIX, the corresponding copy of *lslk* is distributed with A-Shell (or can be obtained from the web). This utility by itself is not that helpful, though, as the information displayed is in terms of inodes rather than files, and process ids rather than jobnames. The **LOKUTL** utility acts as a front end to *lslk* and is able to output a much more interesting display.

Under NT, Windows 2000 and XP Server editions, the Server Manager utility allows display of file locking information, but this is not accessible from within A-Shell.

MAP

```
MAP {/f{ull}}
```

MAP displays the contents of user memory, like its AMOS counterpart. If the /F (or /FULL) switch is specified, it displays the module flag codes associated with each module (from the table below) and also the module version. :

Beginning with version 2.2(106), **MAP** displays the version number of program modules if the /F{ULL} switch specified. This is particularly helpful when trying to figure out if you've got the proper version of an auto-loaded SBX or RUN file in memory.

Flag	Meaning
C	Module is cached in memory and will be deleted automatically as needed.
L	Module is locked (see LOAD / LOCK) and cannot be deleted except by DEL /UNLOCK.
P	Module is "permanent". (AMOS terminology, indicating that it was manually loaded and thus will not be automatically deleted. Use DEL to delete.)
R	Module is running and cannot be deleted under any circumstances.
T	Module is temporary (will be overwritten by next module loaded).

MEMORY

```
MEMORY <size> {k}{m}
```

MEMORY is similar to its AMOS counterpart, except that it can be used at any time to change the user's memory partition size. (Under AMOS you can only reduce your partition size after the system is booted.)

The initial memory size is set by the MEMORY statement in the MIAME.INI file.

Note that although A-Shell's user memory implementation has some similarities to the AMOS equivalent, there are also several differences. For one, when you use **LOAD** to load modules into user memory, they do not actually take up space in the "partition" set by the **MEMORY** command. (Instead, space is dynamically allocated for loaded modules.) This is also true of a number of other memory allocations, such as temporary memory needed to load an SBX subroutine or for use in sorting. Thus, it is likely that you can get by with a smaller partition under A-Shell than you needed under AMOS. (On the other hand, memory is cheap, so why worry about it?)

MMAP

Documentation update June 2004; MMAP 1.0(100) was added to A-Shell in build 851, 05 Oct 03.

```
MMAP {/l}
```

Using the above format, **MMAP** displays (or with /L outputs to MMAP.LST) the current list of files to be automatically memory mapped.

```
MMAP <new mmap fspec>
```

Using the second format, **MMAP** replaces the current memory map list with the one contained in *new mmap fspec*. The format of the list consists of one AMOS filespec per line, the same as for MMAPLIST statement in MIAME.INI file.

MOVE

MOVE <out-wildspec>=<in-wildspec1> {/delete}{/query}

(UNIX only) **MOVE** moves files from one place or name to another. This is similar to copying the file and then erasing the source file, except that it is implemented as a wrapper for the UNIX *mv* command (which is why it only works there). **MOVE** is much more efficient than COPY followed by ERASE, since it moves only the directory entry, not the actual data. Because of this, it will also not update the file modification (DIR /UDATE) date, although it will update the /CDATE. When the source and destination PPNs are the same, **MOVE** is equivalent to RENAME.

The *delete* switch is needed to overwrite an existing output file. The *query* switch causes **MOVE** to prompt for a confirmation before copying each of the input files. <*out-wildspec*> may be a single file, or a wildcard specification. <*in-wildspec*> may be a single file, a wildcard specification, or a comma-delimited list of file or wildcard specifications.

If you like this utility but do not want to worry about it not being available under Windows, you have two choices (besides not using it). One would be to volunteer to modify **MOVE.LIT** to work under Windows. (We would be happy to provide you the source code.) The other would be to create a MOVE.DO for your Windows installations which performed COPY and then ERASE. Since DO file argument parsing is somewhat less flexible than for LITs, you would probably want to require that the equals sign be delimited by spaces on either side so as to make sure in the ERASE statement that you were only erasing the source arguments. The example below illustrates one approach to this, with appropriate syntax checking:

```
;MOVE.DO - Windows approximation of MOVE.LIT
:R
IF "$0" = ""
   GOTO SYNTAX
ENDIF
IF "$1" <> "="
   GOTO SYNTAX
ENDIF

COPY $0 $1 $2 $3 $4 $5 $6 $7 $8 $9
ERASE $2 $3 $4 $5 $6 $7 $8 $9
EXIT

;SYNTAX
:<  Usage: MOVE dest = source
    (Note required space on either side of =)
>
EXIT
```

PAGE

```
PAGE <command line>
```

The **PAGE** command can be used as a prefix to any other (non-screen-oriented) command to causes its output to be displayed one page at a time (instead of scrolling). It stops after each screen-full of output and waits for any key to proceed. For example:

```
PAGE DUMP FILE.TXT
```

This will dump the contents of the file a page at a time. General paging may also be turned on and off by using **PAGE /ON** and **PAGE /OFF**.

This command can be extremely useful when dealing with rapidly outputting commands, such as **TYPE**, **DIR**, **DUMP**, and **COMPIL**.

You can terminate the command by entering *Q* when prompted to proceed to the next page.

PRE/OPRE

Originally, AlphaBASIC did not contain an include file facility, now achieved with **++include**. To get around this problem, a pre-compiler was developed, and is in use by many dealers today (particularly in the UK). This pre-compiler uses **$COPY** to indicate a file to be included in exactly the same way. The difference is that **$COPY** statements may be nested whereas **++include** statements may not. The pre-compiler generates an intermediate file (**.PRE**) which contains the expanded source, and which is then normally compiled automatically and erased. It may be left on disk, however, by specifying the **/F** switch. In addition, any of the compiler line switches may be specified, and these are passed on to the compilation phase.

The pre-compiler was traditionally called **PREBAS** under AMOS. As with the compiler itself, A-Shell contains two versions, one for AlphaBASIC 1.3 (**OPRE**) and one for AlphaBASIC 1.4 (**PRE**). A .**DO** file (with the name **PREBAS** if desired) may be set up to use the appropriate version.

As with **++include**, **$COPY** will accept either AMOS-format file specifications or host pathnames as arguments.

PRINT

PRINT {spooler=}<wildcard list> {switches}

The A-Shell version of **PRINT** functions similarly to the AMOS version and supports the switches given in the table below. As with the AMOS version, you only need to include enough characters of the switch name to specify it uniquely. Thus /NOB is equivalent to /NOBANNER, etc. Also, the default for most switches may be set by a statement (e.g. BANNER = OFF) in the INI for that printer.

If the spooler name is omitted, the default (as specified by the PRINTER= statement in the MIAME.INI will be used. It must match up with a spooler initialization file named %TEMP%\<spooler>.PQI or SYS:<spooler>.INI or ASHCFG:<spooler>.PQI which contains the configuration parameters for that logical printer. Under A-Shell/Windows, you can also use the special spooler name "PROMPT" (which displays a typical Windows printer selection dialog) or you can specify a printer name that is recognizable by Windows (i.e. the type that appear underneath the printer icons). In these last cases, there will be no printer initialization file, and a default configuration consisting of PASSTHROUGH=OFF and PITCH=AUTO will be used.

A-Shell does not include an internal implementation of the spooler, and instead uses the "natural" spooler for the host operating system. Thus, the precise effect of most of the option switches described will vary from one operating system to the next. See the *A-Shell Setup Guide* for more information on printer configuration.

Switch	Meaning
{NO}BANNER	Controls whether a banner is sent at the start of the printout. Under Windows this has no effect. Under most UNIX versions, the default spooler behavior is to force a banner regardless, unless the option to request no banner has been specifically enabled with something like: `# /usr/lib/lpadmin -p <ptr> -onobanner`
COPIES:##	Specify number of copies of file to print.
{NO}DELETE	Controls whether the file is deleted after printing.
{NO}FF	Controls whether a formfeed is sent after the last page.
{NO}HEADER	Controls whether a header is printed at the top of each page indicating the filename, time, date, and page number. Since this feature is not offered by most non-AMOS spoolers, the A-Shell PRINT command has to simulate it by creating a temporary file with the headers and then spooling that file in place of the original.
{NO}LANDSCAPE	(Windows only) Overrides the ORIENTATION parameter in the printer ini file.
LPP:##	When used in conjunction with the /HEADER option, defines the number of lines printed per page.
MARGIN:##	When used in conjunction with the /HEADER option, defines a left margin size.
{NO}PASSTHROUGH	Controls whether the file is sent "raw" to the printer or is allowed to be "cooked" by the printer driver. Refer to the *A-Shell Setup Guide* for more information on this important topic.

Switch	Meaning
PREFIX=fspec	Allows you to specify a file that will be prepended (as a prefix) to the file you are printing (overriding any PREFIX option specified in the printer INI).
QUERY	Standard CMDLIN switch – causes each matching filename to be displayed and the user given a Y/N choice of whether to print that file.
SUFFIX=fspec	Allows you to specify a file that will be appended (as a suffix) to the file you are printing (override any SUFFIX option specified in the printer INI).
WIDTH:##	When used in conjunction with the /HEADER option, defines the position of the "Page #" at the right side of the page header.

The technique for displaying the print queue and canceling a printout varies from one operating system to the next. Under Windows, you just use the Print Manager (generally accessible by double-clicking on the printer icon in the system tray at the end of the task bar). Under Linux, the typical equivalents are lpq and lprm. To execute these from the A-Shell dot prompt, just create a DO file which uses the HOST command (e.g. HOST lpstat –t). Also, most UNIX variations have a menu-driven or even GUI utility for doing these kinds of things.

QUTL

A-Shell on multi-user systems (and even stand-alone) uses a queue file, **qflock.sys**, in which to store information which may be shared by multiple users. This file is in the form of a linked list, analogous to the AMOS queue blocks system, and is used by the **FLOCK** and **XLOCK** subroutines.

The **QUTL** command is primarily intended to give some control over the FLOCK and XLOCK locks present on the system, by reading and processing the queue file. It can display which users currently have locks, and is also able to re-initialize the XLOCK and FLOCK queue lists , a process which normally requires a machine reset under AMOS, or some specially written assembly language routines. **QUTL** may also be used to zap a logged on A-Shell user. This will remove all locks and queue blocks associated with that user, and although the user may continue to be able run whatever program they are in, as soon as it attempts to make an XLOCK or FLOCK call, or when it exits back to the dot prompt, it will see that it has been removed from the queue and will abort with an appropriately (cryptic) error message. (Something like "your queue file has been zapped.")

When used to list the existing XLOCKs and/or FLOCKs, **QUTL** provides an option to output the list to a file. This, combined with the fact that you can run QUTL from a command file, provides a mechanism for nightly processing programs to check if any locks were left in use, and optionally remove them (either by zapping the queue entirely, or just the user, or by using the KILL.LIT utility).

You must be logged into OPR: in order to use the queue utility.

RENAME

```
RENAME <out-wildspec>=<in-wildspec1> {/delete}{/query}
```

RENAME renames files without moving them.

Parameters

out-wildspec
: may be a single file, or a wildcard specification.

in-wildspec
: may be a single file, a wildcard specification, or a comma-delimited list of file or wildcard specifications.

delete
: is needed to overwrite the name of an existing output file (whose contents would then be deleted).

query
: causes **RENAME** to prompt for a confirmation before renaming each of the input files.

Although **RENAME** appears to have the same syntax as

COPY(and MOVE), it is restricted by the fact that the output file location must match the input file location. This is normally used on files in the current directory, eliminating the need to specify a device or PPN.

SAVE

```
SAVE <name.ext> {/delete}
```

SAVE, like the AMOS counterpart, saves a module from user memory to disk. The */delete* switch is needed to overwrite an existing file.

SEND

SEND (as of A-Shell 4.6) applies to both Windows and UNIX, and although it operates differently in each case, the general idea is pretty much just like the AMOS equivalent.

```
SEND {switches} <target name> <message>
```

The two optional switches are /T and /B, which cause the message to appear on the top or bottom status line, respectively, of the target terminal, rather than at the current cursor location. This assumes that the target terminal supports status line operations; otherwise the switch is ignored.

The <*target name*> may be either the job name or the login user name of the person you want to send a message to, or "ALL", "TODOS", or "TOUS" for everyone. (You can use the SYSTAT command to help you determine which job and user names are available.)

Under UNIX, **SEND** is implemented via the operating system's signal delivery system. If the target is running PolyShell, then the message will appear in a pop-up window. Otherwise, the message will simply appear at the current screen cursor location, overwriting whatever was there previously.

> Because of UNIX security rules, it is impossible for one user to send signals to another unless the first user has superuser privileges, or unless they are both sharing the same effective user ID. Refer to the *A-Shell Setup Guide* (Installation) for instructions on achieving this.

Under Windows, **SEND** is implemented using the "Inter Job Communication" (IJC) system, which was introduced in A-Shell 4.6. Unless the /T or /B switch is specified, the message is displayed in an independent pop-up window using the AWMSG.EXE utility which is distributed with A-Shell. The message window initially appears on top of all other windows, and does not interfere with the target A-Shell window in any way.

SET

The **SET** command under A-Shell supports a subset of the AMOS display and settings, as well as some extensions. The functions that are fully supported are listed below, followed by notes about features that differ significantly from the AMOS counterpart. Entering **SET** with no specifications will display current settings, and **SET/?** will display options. The items marked below with an asterisk have no corresponding AMOS equivalent.

Function	Displays
{NO} ABSLOOKUP *	Always return absolute value in LOOKUP.
{NO} AHASH *	Turn on/off AMOS compatible hash codes.
{NO} AMOSRUNSBR	Execute AMOS.SBR within current process.
{NO} AUTOTPRINT	
{NO} CMDINP *	CMD file input.
{NO} CTRLC	Control-C.
{NO}DEBUG {#}	Cause RUN programs compiled with /X:2 but not /DEBUG, to interpret the DEBUG variable as 1 (or # if specified.)
{NO} ECHO	Character echo.
{NO} EXTFIO *	Extended file I/O.
{NO} FIELDEMU *	Field Emulation.
{NO} FPROUND *	Floating point expression rounding.
HEX	Set Hex output.
INFDEF	Change or display INFLD codes.
{NO} LATIN1 *	LATIN1 8 bit mode.
{NO} LOKSER *	Turn LOKSER on/off.
LONGDIR	Forces DIR to act as if the /L{ONG} switch had been set.
OCTAL	Set Octal output (e.g. in DUMP).
PROMPT <prompt>	Set command prompt to <prompt>.
{NO} STRICT	Strict AMOS compatibility mode.
TERMINAL options	Terminal settings; see sub-commands below.
TITLE definition string	Set top status line or window title.
TRACE options	Various trace options; see sub-command below.

AHASH

This turns on and off AMOS compatible hash codes, and applies only to text files under A-Shell/UNIX. The default is off. When on, the A-Shell hash routine will pretend as if there were a CR to go along with any LF line terminators when generating a hash code (since UNIX text files normally only have LF line terminators whereas AMOS, Windows, and DOS use CRLF).

CMDINP

This turns on and off command file input within XCALL INPUT or XCALL INFLD. The default is off, in which case these input routines do not accept input from a command file. INFLD supports a TYPE code "p" which turns on command file input, and you can force this on by adding SBR=INFDEF:p to your miame.ini file. But these methods either require a change to your program or that you exit from A-Shell and modify the miame.ini file. The **SET** {NO}CMDINP command, on the other hand, can be used within A-Shell, or even within a command file. For example, you might have a command file which contains:

```
:R
SET CMDINP
RUN MENU
15
<program inputs>
SET NOCMDINP
RUN MENU
HOST
```

The above command file starts by setting command input on, and then runs a menu program, and forces it to select option 15. It then continues to include the necessary prompts for the program corresponding to menu option 15, after which that program finishes and returns to the command file. Then it turns command file input back off again and starts up the main menu, which now waits for real keyboard input. Finally, when the user exits out of that menu, the command file picks up and forces A-Shell to exit.

INFDEF

This was added to SET.LIT in version 1.0(142), of 6 January 2005.

SET.LIT now supports an INFDEF option to display or change the list of default INFLD codes (previously only able to be set via the miame.ini SBR=INFDEF statement.) Syntax:

```
.SET INFDEF             (display)
.SET INFDEF <codes>     (replace current defaults with <codes>)
```

xcall MIAMEX,143,OP,INFDEF allows the same capability from within a program. (MX'INFDEF is defined as 143 in ashell.bsi edit 103). OP is a numeric argument; if 0, the current default type codes are returned in the INFDEF parameter, otherwise the current default type codes are replaced by those in the INFDEF string.

The maximum size of the default string is currently 36.

This feature is of course handy for experimenting with global INFLD TYPE codes within a single job (without interfering with other jobs). It may also be useful for changing the defaults with a single program. Note that for a particular INFLD call, you can disable the default TYPE codes by adding m to the TYPE parameter passed to INFLD.

LOKSER

This turns on and off LOKSER. This provides the same functionality as the LOKSER=ON/OFF statement in the miame.ini file but can be used without exiting and re-entering A-Shell (as would be required to modify **miame.ini**).

STRICT

The STRICT function was added to A-Shell in build 859.4, 14 Jan 2004.

SET.LIT 1.2(135) allows you to turn the STRICT option on and off. This may be convenient in CMD files if you only want STRICT mode for certain operations.

TERMINAL options

A-Shell/Windows allows you to set the terminal height to any value from 1 to 50 rows, and the width to any value from 1 to 160 columns. (This can also be done via the TAB(-5,rows) and TAB(-6,cols) commands.) Under UNIX, the ability to do this is limited by your terminal emulator. In most cases you are limited to 24 rows and either 80 or 132 columns. The notable exception is ZTERM, which in the most recent updates supports variable rows up to 50. Typing "SET TERM" by itself will display the ZTERM version, if applicable, in addition to the current width, height, foreground and background colors and terminal driver.

Variable	Values	Description
FCOLOR n	n	Set foreground color to n
BCOLOR n	n	Set background color to n
WIDTH	80 or 132	Set terminal width to 80 or 132 columns
BEVEL	off, auto, program	Set beveling option to off, auto or program
SYSBCOLOR n	n	Link background color n to Windows dialog/button color (gray)
WINBCOLOR n	n	Link background color n to Windows edit box color (white)
WINTXTCLR n	n	Link foreground color n to Windows text color (black)
TSTATUS	on / off	Turn top status line on/off
BSTATUS	on / off	Turn bottom status line on/off

Comments

As of Build 897, 10 August 2004, SET.LIT supports two new commands to link A-Shell color palette numbers to Windows desktop colors, to go along with the existing SET TERM :

```
SET TERM WINBCOLOR n
SET TERM WINTXTCLR n
```

The first (WINBCOLOR) sets the "Window Background" color n, which is typically used for the background of small windows such as edit boxes, checkboxes, and listboxes (typically white).

The second (WINTXTCLR) sets the "Window Text" color n, which is typically used for text both on buttons and in edit boxes and listboxes (typically black). Note that the n here is taken from the foreground palette (unlike WINBCOLOR and SYSBCOLOR which both refer to color numbers from the background palette).

Along with SYSBCOLOR, these can now be used via ATE on UNIX servers.

Added in Build 904, 13 October 2004: TSTATUS and BSTATUS may be set <on / off> to control the presence of the top and bottom status bars. Turning the status bars off makes for a somewhat "cleaner" display when doing screen captures for demo, slide shows, documentation, etc.

TITLE definition string

The SET TITLE command takes a string argument, optionally preceded by $TS, and which may contain any literal text that you want to appear as the title. If the $TS prefix is specified, the title will appear on the top status line; otherwise it will appear on the window title bar, providing you are running A-Shell/Windows, or you are using ZTERM as your terminal emulator to connect to some flavor of A-Shell/UNIX. The following special variables may appear anywhere in the title string:

Variable	Displays
$NC	Name of last executed command (e.g. "SET", "LOG") or program.
$ND	Name of current logged in device (e.g. "DSK0")
$NJ	Your job name
$NP	Name of last program executed (same as $NC but excluding LITs)
$PA	Platform A-Shell was compiled for (e.g. "Linux", "Windows/32", etc).
$PN	Current p,pn (e.g. "[7,6]")
$VA	Version of A-Shell (e.g. "4.6(753)")
$VP	Version of current program or command (e.g. "1.0(123)")

If the title definition string contains any of the above variables, the title will be automatically updated when the value of that variable changes. For example, if the title is defined via:

```
SET TITLE Job $NJ running $NP version $VP in $ND:$PN
```

Then whenever the program name (excluding LITs) or the current logged in device or PPN changes, the title line will be updated, and would appear something like this:

```
Job TSKAAB running MYPROG version 1.0(123) in DSK0:[7,6]
```

If you do not specify the $TS prefix (for top status line) and you are not running on either A-Shell/Windows or using ZTERM as your emulator, then the title will just be ignored.

If you do specify the $TS prefix and you are running PolyShell, then the title on the top status line will be updated whenever you switch to that window. If you have job-specific information in the title string, then it is highly recommended that you use SET TITLE on each of the processes running under PolyShell; otherwise, when switching to a job or session with no title definition, the previously displayed title will remain, possibly causing confusion for the user.

TRACE options

These are equivalent to the identically named TRACE options in the **miame.ini** file. Since they are primarily used only for debugging, it is a handy convenience to be able to turn them on temporarily for a single job

without having to modify the **miame.ini** file and re-launch the session. These variables accept only on or off settings.

Note that depending on the operating system you are running, some of the options may not appear. (For example, signals are only available under UNIX.)

In most cases the output of the traces go to the ashlog.log file (in the same directory as miame.ini, unless specified elsewhere with the LOGDIR= command in the miame.ini). Traces that go to the screen are noted specifically in the table below.

Note that any attempt to set any other parameter, even if it is a valid parameter under AMOS, will result in the same error message ("?Invalid function").

Trace	Notes
AMOS	Logs information related to the use of XCALL AMOS and operations that involve the launching of another ashell session.
AMSORT	Displays sort information.
ATE	Logs details of the GUI commands and responses.
BASERR	Logs all Basic errors, indicating if they were trapped or not.
DEBUG	This is a catch all for logging all kinds of information. It should only be used in focused circumstances, since the amount of tracing data quickly becomes overwhelming.
FOPENS	Logs information about file opens (both programs and data). Under Windows, this also includes how many microseconds it takes to open the file, which can sometimes be useful in identifying bottlenecks due to network issues.
GDIPRT	(Windows only) Logs information about processing of GDI printing directives.
GUI	(Windows only) Logs information about TAB(-10,x) commands and operations on GUI objects.
ISAM	Logs all ISAM statements
LOCKS	Logs details of locking operations, and in some cases, displays (on the status line) when the program is waiting on a lock.
LP	Displays details about printing.
MALLOC	Log details of all memory allocations
RW	Log all random/isam file reads and writes
SIGHUP	Log receipt of SIGHUP (hangup) and SIGTERM (kill).
SIGNAL	(UNIX only) Display received signals on screen (e.g. “[SIGINT]”, “[SIGALRM]”, etc.)
SOCKS	Log details of socket operations (TCPX.SBR or ATE control socket)
SYSERR	Display details about all errors.
USRMEM	Log details about “user memory” operations – i.e. loading of programs, subroutines, function key translation files, etc.
XCALL	Log all XCALLs (showing subroutine name and parameter list.) Often useful as a way of tracing what a program is doing or where it was when an unexplained error occurred.

AMSORT

Displays information (in a pop-up window) during sorting operations, showing the type of sort, current phase, memory usage, etc. If sorts are talking a long time, this can be very useful for determining what is happening, and how much memory you might need to improve the performance. (For very large sorts, we also offer an add-on "world class" sort module that is much faster and more flexible, but requires a small license fee; contact us for details.)

ATE

Logs details of the GUI commands and responses sent between a server and an ATE client. Note that this command might make sense to activate on both the server and the client, as they have different perspectives. You will have two log files though – one on the server and on the ATE client.

LP

Displays on the screen details about printing, including processing of any EZSPL configuration file, plus the printer init commands, plus the actual communication or initialization operations with the printer itself. Extremely useful for debugging printing problems. (SET TRACE LP ON, then using PRINT to print a dummy file, like ERSATZ.INI).

SIGHUP

(UNIX only) Log receipt of SIGHUP (hangup) and SIGTERM (kill) signals, along with information indicating how the signal was responded to, whether the program and ashell session were shut down, etc. We recommend you set this switch permanently in the miame.ini file (TRACE=SIGHUP) since they should be relatively rare and frequently either worth investigating directly, or they provide an explanation for something else you are investigating.

SYSERR

Display (in a pop-up window on screen) details about all errors. This is sometimes useful for see the system error details corresponding to a particular A-Shell error, but can also be annoying since it presents the appearance of a serious error having occurred, sometimes in situations that weren't really very serious at all.

SRCCOM

Documentation update June 2004; see "Update Notes" below.

```
SRCCOM {listfile=} oldfile, newfile {/switches}
```

SRCCOM compares two source files, reporting the differences. This is useful, for example, when comparing two versions of a program to see what has changed between them. It is most useful when the differences between the files are few; when they are substantial, it becomes both overwhelmed and overwhelming.

Switch	Effect
B	Brief listing – same as /Q but without line numbers
Q	Quick listing – list only the lines that differ between the two files

Under UNIX, there is a much more powerful file comparison utility call *diff*, and in some cases, *diff3*, which can compare three files at a time. See the notes on DIFF, a front-end to *diff*.

Update Notes: Version 1.0(108), Build 868.1, 25 Feb 04

SRCCOM.LIT now includes the list file name and switches in the listing header (matching the format of the AMOS version).

SUBMIT

To submit a task:

```
SUBMIT ctlfile {do-params} {/i inifile} {/l logfile} {/ba buffile}
     {/bc buffile} {/j job} {/w} {/next:timspc} {/after:timspc}
```

To kill a task:

```
SUBMIT /K seqno | pidno | jobno
```

To list tasks submitted by this job that are currently running:

```
SUBMIT
```

The A-Shell **SUBMIT** command is similar to the AMOS counterpart, in that it causes a new process to be launched, either now or at a specified later time, which executes commands specified in the control file. Under AMOS, this is managed by a background job called the Task Manager. Under A-Shell, there are actually three different possibilities, depending on the operating system and whether the */after* and */next* switches are used:

- Windows
- UNIX without switches
- UNIX with switches

In all cases, like its AMOS counterpart, **SUBMIT** requires a control file (typically with a CTL extension) which functions more or less just like a command or DO file (i.e. containing the commands necessary to run the job, along with any keyboard inputs.) Programs running under the control of SUBMIT will automatically take their input from the control file, even if they wouldn't normally accept input from a command file. When the control file runs out of input, the process terminates. (The preferred way to terminate the process is for the program to cleanly exit back to the control file, and for the control file to end, optionally with a LOGOFF command.)

Switch	Effect
Do file parameter	Unlike the AMOS implementation, the A-Shell version of SUBMIT allows DO file parameters to be passed to the control file (which can use the $0-$9 syntax, just like normal DO files, to represent the formal parameters). Any space-delimited tokens following the control file name and preceding the first switch (marked by a slash) will be passed to the control file as DO parameters.
/I "inifile"	Forces the child process to use a different A-Shell configuration file than the parent process. Note that the inifile name should be a native file specification, and must be enclosed in quotes (to prevent the slashes in the file spec from being confused with switches).
/L logfile	(UNIX only) Overrides the default log file name. (Specify an AMOS-type filename.)

Switch	Effect
/J jobname	Functions just like the **-j** switch to A-Shell (allows you to give the background process a specific name). Otherwise, a name of the form TASxxx (TASAAA, TASAAB, ...). will be generated automatically.
/BA buffile	Activates the auto-snapshot feature of INFLD and appends each snapshot to the specified file. The default file extension is BUF, and the default location is MEM0:[1,1] (if it exists) or the current login directory where the first snapshot occurs. Works internally by passing the –ba <buffile> switch to the child A-Shell process and by adding the |S type code to the INFDEF (default INFLD parameter) string.
/BC buffile	Same as the /BA option except creates a new snapshot buffer file (overwriting any existing one).
/W	(UNIX only) Wait for the submitted process to complete. This may seem to defeat the purpose of running a process in background, but can be useful either when the main objective is to look at the LOG file, or when you want to run processes that otherwise would not accept input from a command file.
/K seqno|pidno|job	(UNIX only) Kill a process currently running. To identify the process, you may specify either the "sequence number", the "process id (pid) number", or the job name (which you can get by using the SUBMIT display mode, with no arguments). Note that unless you are the superuser, you may not be able to kill any background processes other than those that you have launched.
/AFTER:{+}timspc	(UNIX only) Causes the control file to be submitted via the UNIX 'at' background processor, instead of as a child to the current job. (See note above about the 'at' processor.) The timspc argument may be specified using the form M-D-Y{,HH:MM}. The M-D-Y portion must be specified simply as 0 for today. The optional {+} makes the time specification relative to now. Examples: /AFTER:0,23:00 (11 PM tonight) /AFTER:06-30-03,00:30 (12:30 AM on June 30, 2003) /AFTER:+1-0-0 (1 month from now).
/NEXT:timspc	(UNIX only) Equivalent to /AFTER:+timspc.

Windows

Here, **SUBMIT** only partially simulates the Task Manager by launching a new instance of A-Shell to run the specified task. This is acceptable for tasks which have explicit outputs or side effects (for example, a report or series of COPY commands), but it does not really run in "background", and there is no LOG file. In the Windows environment, if you want to schedule a task that is launched by the operating system at the specified time, you should just use the Windows Task Scheduler (Programs | Accessories | System Tools | Scheduled Tasks).

UNIX without switches

In the case of A-Shell/UNIX without the */after* or */next* switches, A-Shell launches a new process to run the specified task as a child of the current process. However, unlike the Windows case, the new process does run in background, with all of its screen output being written to the LOG file (e.g. a file with the same name and location as the *ctlfile*, but with a LOG extension.) If the parent exits before the child does, the child task will continue to run but will be marked as a zombie.

In the above case, if there is an error during the attempt to launch the new instance of A-Shell (perhaps because the ashell executable is not in the PATH or the MIAME environment variable is not defined), then information about the error will be written to the standard A-Shell log file (ashlog.log, in the directory where the miame.ini is).

UNIX with switches

The case of A-Shell/UNIX with the */after* or */next* switches is the most similar to the AMOS Task Manager. The task is submitted to the UNIX at manager, which, like the AMOS Task Manager, keeps track of it in a queue until it launches it at the specified time. The process will run with the same login identity and environment as the process that submits it, but since it runs as a child to the at manager, it is independent of the original submitter (and thus not subject to becoming a zombie.) As in the prior case, all output that would otherwise have gone to the screen is sent to the LOG file instead.

The **at** manager is similar to the more familiar **cron** manager, both of which can be found on almost any UNIX/Linux system. The **cron** manager is more suited to tasks that are to be run at regularly scheduled intervals; you submit jobs to it via the **crontab** command. The **at** manager, on the other hand, is more suited to jobs that are to be run once at a specified time in the (immediate or distant) future. To submit jobs to be run by the **at** manager, you can use the UNIX at command. SUBMIT.LIT just acts as a front-end to **at** to allow you to use the simpler AMOS command syntax.

SYSTAT

Documentation update November 2004 and January 2005; see "I" and "M" switches below.

The A-Shell **SYSTAT** command is very similar to the AMOS counterpart, and produces a display something like this:

```
Status of A-Shell/32 Version 4.9(927) on Monday, June 06, 2005 10:25:33 PM
TSKAAA TSKAAA  jack            DSK0:7,6       RN  SYSTAT   62343  400K bytes
TSKAAB TSKAAB  jack            DSK0:1,4       ^C  SET      66145  400K bytes
2 jobs allocated on system, 2 in use
DSK0      36592 Blocks free     DSK1      36592 Blocks free
DSK2      36592 Blocks free     DSK3         -1 Blocks free
4 devices on system, total free blocks may be shared among devices
```

The lines which correspond to the individual jobs are actually formatted identically to the AMOS version of **SYSTAT**, except that the memory size is always shown in K, to make room for the UNIX process id, which displays (under A-Shell/UNIX) between the program name and the memory size. The available job states are currently only "RN" and "^C", unlike the AMOS version which shows many more states.

Note that in the list of devices, one or more of the pseudo AMOS-like devices may share the same physical disk device on the host operating system, and thus the number of blocks free is the same. In the example above, DSK0, DSK1, and DSK2 are all hosted on the same device, while DSK3 is apparently not accessible.

Switches

The A-Shell version of **SYSTAT** offers a number of interesting switches which can be listed with **SYSTAT/?**

Switch	Effect
A	Displays all of the "allocated" jobs (i.e. those listed in TRMDEF.INI) even if no corresponding process has been launched.
B	Displays only background jobs.
C	Displays the controlling terminal's device identification.
F	Displays only foreground jobs (i.e., jobs with display devices attached).
I	Displays IP address. This feature added in SYSTAT 3.0(150).
K	Checks for and kills phantom jobs under UNIX.
L	Sends the output of the display to the file SYSTAT.LST in the current directory rather than to the screen. This is useful when you want to examine the state of the users programmatically.
M	Displays the minimum amount of free (unused) memory so far in the life of each job, instead of the allocated amount. This can be useful in judging whether your memory partitions are reasonably sized.
N	Eliminates the display of the devices. This is particularly useful when you have a lot of devices, especially if some of them are slow to respond due to being offline or connected over a WAN.

Switch	Effect
P	Pause after each page of display. (Also see the notes on PAGE for another way to get the same effect.)
R	Display “Real” jobs only. Omits PolyShell control jobs from the display.
S	Sort the display of users by job name.
SU	Sort the display of users by user name.
W	Display in wide mode. (Also totals several of the columns)
Z	Similar to /K but for “zombies” rather than “phantoms”.
/ZS	Identical to /Z except that it skips zombies that were launched via SUBMIT. Such jobs predictably become zombies if the submitting job exits, even though they may continue to function as intended. Thus you may not necessarily want to kill them along with other zombies.

C

Displays the controlling terminal’s device identification rather than the user’s name. Under UNIX, the user’s name is set by the UNIX login, whereas the device name is determined by the user’s connection to the host. Under Windows, the user name and device name are the same except for a numeric suffix, since both come from the NETBIOS machine name.

K

Checks for and kills phantom jobs under UNIX. These can occur when the underlying UNIX process is somehow destroyed (such as by a **kill –KILL** operation performed by a superuser) without A-Shell being able to trap and recognize it. If such a process is found, SYSTAT will then remove it cleanly from the job table, and output both an asterisk in column 7 of the display, as well as a more descriptive message on the following line.

W

Display in wide (132) column mode (if terminal supports it). This allows the display of additional job status information, including the number of disk reads, writes, queue locking operations, commands (or programs) executed, BASIC instructions executed, keystrokes entered, and accumulated time logged in. Note that these statistics are only updated by the Inter Job Communication clock, whose frequency is set by the IJCFREQ=<seconds> command in the MIAME.INI.

Z

Similar to /K but for “zombies” rather than “phantoms”. A zombie is a job that is trying to exit but cannot because it lacks some kind of critical resource (like a terminal or its original parent). Unlike phantoms, zombies can be a significant drain on the CPU resources of the system. In most cases, zombies are related to some kind of messy telnet disconnection. (Activate TRACE=SIGHUP in the miame.ini or using SET.LIT and then examine the ashlog.log file for further information about such disconnects.)

TAIL

```
TAIL {outfile=} <wildspec> {/switches}
```

The **TAIL** command is similar to the eponymous UNIX command and is used to display or copy to another file the last N lines of a specified file (or wildcard list of files).

Switches

delete
Required when an output file is specified that already exists.

lines:##
Specify the number of lines to copy or display. (Default is 20).

query
Prompt for confirmation on each file.

See the notes on HEAD for examples.

TELNET

```
TELNET <connection>
```

(Windows only) **TELNET** launches ATE (the A-Shell Terminal Emulator) to establish a session with the specified connection. The connection name must have been previously defined using ATECFG (which see).

You can terminate the session either from the remote host (i.e. by logging out) or by entering "|D" (vertical bar, D).

TELSER

```
TELSER <port#> <startup-command> {/switches}
```

(Windows only) **TELSER** allows you to turn your current A-Shell/Windows session into a single-client telnet server. That is, it proceeds to listen on the specified port for a standard telnet client connection, at which point, the remote telnet client takes over the session. This is a rather advanced (or perhaps obscure) capability, which is mainly of use for remote support (sort of like pcAnywhere) or for environments where your application was written to depend on the capabilities of a particular terminal emulator.

Switches

s
: Display waiting window on task bar (otherwise it becomes invisible)

T:tdv
: Forces the client connection to use the specified terminal driver. Otherwise it will negotiate the terminal type with the client terminal emulator.

TYPE

```
TYPE <filespec> {/switches}
```

TYPE displays the specified file on the screen (similar to the *cat* utility under UNIX).

TYPE <file>/P is functionally equivalent to PAGE TYPE <file>; see the notes on PAGE.

TYPE will display control characters in "caret notation" (i.e., using a printable character preceded by a ^). For example, nulls will display as ^@, escapes will display as ^[, etc.

Switches

p
: Page mode: stops after each screen page

s
: Slow mode: displays only 10 lines per second

U

U simply takes the command line which follows it and writes it to a command file called SYS:U.CMD, so that it can be easily recalled later just by typing U. For example:

```
.U dir/udate/cdate/adate *.bas[],*.run[*,0]
```

After executing the above command, you would then be able to enter just **U** <Enter> to execute the command:

```
dir/udate/cdate/adate *.bas[],*.run[*,0]
```

Since recording one command is of limited value (given that you can use ^R to retrieve previously typed commands from the command history buffer), you can make unlimited copies of the U.LIT command under different names, which will work the same way but which will record their command lines in the correspondingly named command files.

Example

```
.COPY U2=SYS:U.LIT
.U2 DIR/UDATE/CDATE/ADATE
```

The above would create a command file called U2 which could be later executed to execute the DIR/UDATE/CDATE/ADATE command line.

VERIFY

VERIFY <dir-file-spec> {/F}

The **VERIFY** utility reads a previously created directory listing and compares it to the actual files, reporting whether they agree or differ. This is a standard and very useful technique for verifying that a particular installation of files is up to date or not corrupted. (It seems amazing that there is no equivalent utility packaged with Windows or UNIX, although the RPM package manager of Linux does have a similar capability.

The */f* switch outputs the results to the file VERIFY.LST; otherwise it is displayed to the screen.

The specified directory file must have been created with the DIR.LIT /D/V/H switches. For example, when building a release of A-Shell, we execute the following commands:

```
.dir ashell.dir=dsk0:[]/d/v/h/k/sname
```

(The */k* switch is needed to overwrite a previous copy of ashell.dir, and the */sname* switch adds the optional nicety of listing the files sorted by name.)

The resulting directory listing looks something the example below, with each line containing the complete filespec of a single file, followed by the version number and the hash code:

```
DSK0:ABOUT.LIT[1,4]           1.0(103)          656-123-417-74
DSK0:APPEND.LIT[1,4]          2.1(111)          352-256-456-10
DSK0:ASTAT.LIT[1,4]           2.2(131)          121-772-424-00
DSK0:ATECFG.LIT[1,4]          1.0(100)          260-132-704-07
DSK0:C.LIT[1,4]               1.0(100)          167-604-517-71
DSK0:CHAT.LIT[1,4]            2.2(118)          632-367-207-57
DSK0:CLS.LIT[1,4]             1.0(100)          167-604-517-71
```

You may use **VERIFY** on this file to check if the standard files that come with A-Shell are still in sync with the release versions:

```
.VERIFY OPR:ASHELL.DIR
```

The output looks something like this (hopefully):

```
DSK0:ABOUT.LIT[1,4]           VERIFIED
DSK0:APPEND.LIT[1,4]          VERIFIED
DSK0:ASTAT.LIT[1,4]           VERIFIED
DSK0:ATECFG.LIT[1,4]          VERIFIED
DSK0:C.LIT[1,4]               VERIFIED
DSK0:CHAT.LIT[1,4]            VERIFIED
DSK0:CLS.LIT[1,4]             VERIFIED
```

If the version or hash of the actual file does not match that recorded in the directory listing, it will report the discrepancy, for example:

```
DSK0:DIR.LIT[1,4]   ?Version mismatch  is - 1.5(125)  MDL = 1.5(124)
```

This would indicate that the actual version of the file DIR.LIT is 1.5(125) while the release version as recorded in the directory listing is 1.5(124). (MDL stands for Master Directory Listing)

VERSYS

Documentation update June 2004; see "Update Notes" below.

VERSYS <wildspec>

The **VERSYS** displays version and resource information associated with an A-Shell LIT program that relates to the environment in which it was created. For example:

```
.VERSYS DIR
DIR.LIT -- A-Shell Version & System Information:
   File Version:    1.5(125)
   System Resource: ashell[74]
   System Resource: cmdlin[24]: max devs = 300, max ppns = 512
```

The file version can also be gotten from the DIR/V command. But DIR/V cannot tell you the details about the ashell and cmdlin resource files, which were used to compile this copy of DIR. Periodically we update these resource files, which when used to re-generate the LIT commands, may create subtle differences which are not reflected in the version number.

Update Notes: Version 1.0(102), Build 857.2, 23 Dec 03

VERSYS.LIT 1.0(102) now shows the program header flags for RUN and LIT files (which indicate how the program was compiled and whether it is compatible with AMOS). Note that it cannot distinguish whether /A was used when compiling (although it can distinguish /AV, which implies /A), and it cannot tell between /X:1, /X:2 and /D, all of which show as /X and are incompatible with AMOS). This version of **VERSYS** is backwards compatible with A-Shell 4.8.

VUE

Documentation update October 2004; see Special Commands below.

```
VUE <file> {/switches}
```

A-Shell contains an implementation of **VUE** with a similar look, feel, and functionality to AlphaVue version 3.0 under AMOS. Most basic commands and display mode features are implemented, although the emphasis is on general utility and not on being a perfect emulation of the AMOS version. A number of "power-user" extended features have been added, however, to accommodate some of our more demanding users. These are discussed, following the basics, under *Special Commands* below.

Switch	Effect of Switch
C	Enable command file input (including in display mode). Characters are input from the current command file and interpreted as if they were typed. Note that it is critical to know which mode the session will start in (display vs. command) since the interpretation of the commands would differ between the modes. See the START command in the ini.vue file for info on starting in a specified mode.
R	Read-only mode. (You might want to consider EZTYP as an alternative to this.)
T	Trace steps involved in locating and processing the ini.vue file. Useful for debugging.
W	Start in 132 column (wide) mode.
Y	Automatically answer affirmatively to the question about whether to create a non-existent file.

The **HELP** command is fully implemented. Use >**HELP VUE** (from A-Shell/VUE command mode) to get on-screen lists of commands available within A-Shell/VUE. Other useful HELP files shipped with A-Shell are INFLD (e.g. **>HELP INFLD**), INMEMO, TCRT, and SPOOL. (And you may transfer any other HLP:*.HLV files from AMOS to A-Shell for the same purpose.)

INI.VUE

As with the AMOS version, A-Shell's **VUE** starts by searching for an INI.VUE file. The search path consists of the current PPN, the [p,0] directory, and DSK0:[7,0]. If found, the contents are processed in much the same way they would be if they were typed in VUE's command mode. The set of possible commands is shown below. Default values are shown first. Most of the options work under A-Shell as they do under AMOS, with a few exceptions which are noted here.

Command	Values
AUTODELETE	= OFF/ON/#
AUTOSAVE	= ##
CHARINS	= OFF/ON
COLOFFSET	= integer of starting column
COLUMN	= OFF/ON
CVTWARN	= ON/OFF
DEFAULT	= ext1, ext2, ext3, ... (up to 10)
DRAW	= OFF/ON
GO	= COMPIL %
PRINT %.@	
$	
INSERT	= OFF/ON
MODEM	= OFF/ON
NARROW	Forces VUE to start in the normal 80 column mode.
QUITWARN	= OFF/ON
SEARCHFOLD	= ON/OFF
SHORT	Forces VUE to start in the normal 24 row mode.
START	= DISPLAY/COMMAND/END
TAB	= # (default 8)
TALL	{##} Forces VUE to start in a display mode consisting of the specified number of rows (or 42 if no argument given).
UPDATE	= OFF/ON/<sc>{,<ec>}
UPDSSL	= OFF/ON
WIDE	Forces VUE to start in the normal 80 column mode.
WIDTH	##
WRAP	= OFF/ON

AUTODELETE

The **AUTODELETE** and **UPDATE** commands (also available from Command Mode) are similar to the "d/VUE" commands of the same names. The both relate to an optional feature in which an automatic update notation is added to the top of the file whenever you finish out of it. The notation looks something like this:

```
!*! Updated on 02-Oct-98 at 06:33 PM by Jack; edit time: 0:45:23
```

Appropriate default comment markers ("!*!", "/*" and "*/", or ";*;") are determined on the basis of the file extension. If the file extension is not recognized, the feature is turned off. If you prefer to define your own comment markers, then use the **UPDATE** = <sc>{,<ec>} syntax (where <sc> is the "start comment" sequence and <ec> is the optional "end comment" sequence).

When the **UPDATE** feature is active, **AUTODELETE** may come in handy to automatically remove older entries (so you do not accumulate an infinite number of them or have to manually remove them). Setting it to ON results in just the last entry being preserved. If you specify a number instead, it will save that many entries before deleting older ones. See the notes on Special Commands for information about the related ^SE and ^_E commands.

AUTOSAVE

This causes the file to be automatically saved after the specified number of changes are made during an edit session. Single-character operations are counted each as one change; more complicated operations (e.g. replace, line and word delete, block operations, etc.) are arbitrarily counted as five changes. A reasonable number might be something in the vicinity of 1000. A value of 0 disables the operation. (To aggravate your mates and waste a lot of resources, you can set the value to 1.) When triggered, the file is saved exactly as if a manual SAVE command had been issued. Note that even after several such SAVE operations, you can still recover your original file by using QUIT to exit and then making a copy of the .BAK file. (e.g. .COPY MYFILE.OLD=MYFILE.BAK)

COLOFFSET

The COLOFFSET command (both at the command level and in INI.VUE) adjusts the starting column number. By default, VUE calls the first column #0. If you want the first column to be numbered 1, then use >COLOFFSET 1 at the VUE prompt or add COLOFFSET = 1 to INI.VUE. This was added to A-Shell in Build 938 of 25 August 2005.

CVTWARN

This controls whether you get a warning that it will convert the CRLF line terminators (AMOS/Windows format) to native format terminators (LF for UNIX).

DRAW

This controls whether you can invoke a d/VUE-inspired feature of the same name in which the control commands ^S^H, ^S^J, ^S^K and ^S^L can be used to change the default cursor advance direction to left, down, up and right (default), respectively. This feature can be handy for laying out forms, but if accidentally invoked could wreak havoc on a blind typist, which is why it is not enabled by default.

GO

This command specifies the action or sequence of commands to execute when the file is exited with the **G** or **GO** command. The sequence starts with the first command following the equals sign, and continues until the **$** is encountered. The special character **%** may be used to refer to the name of the file being edited (without any extension) while the **@** character refers to the file's extension.

Under the standalone implementation of VUE, the GO command must be limited to a single line (e.g. GO=compil %$)

MODEM

This command (also available from Command Mode) toggles between the normal display mode and a mode in which the amount of output characters is minimized. This mainly affects the way Command Mode works. In standard mode (MODEM = OFF) the entire screen is cleared and repainted when switching between Display and Command Modes. When MODEM = ON, only the top one or two lines of the screen are used for Command Mode, leaving the rest of the Display Mode screen display intact. This abbreviated command mode is particularly appreciated by modem (or other slow baud rate), as well as people who tend to forget what they were doing when the screen is suddenly cleared.

QUITWARN

This command controls whether you get a warning message and request to confirm when you attempt to quit out of an editing session after making changes to the file. (Of course if you still quit accidentally, you can always retrieve the .BAK backup file, but you must do that before you edit the file again.)

UPDSSL

This may be set to ON to force VUE to keep the shifted status line (if there is one) equal to the unshifted status line. (This prevents the line from "blinking" when you hit the SHIFT key.) This is only an issue in certain terminal emulation environments.

Function Keys

The MIAME environment contains full function key support, but these will only be usable within the editor if the appropriate function key translation files exist in account DSK0:[7,0]. These are files with the same name as defined for your terminal in the TRMDEF line of the MIAME configuration file, and extension .VUX.

PCTDV.VUX and PCTDVG.VUX are supplied with A-Shell/Windows, either of which may be copied or renamed if your A-Shell/Windows terminal driver is going by a different name. Under A-Shell/UNIX, there are several available terminal drivers (am62a, am62c, am65, am75, wyse50, etc.), each of which needs its own VUX file.

The **FIXTRN** utility may be used to create, query or edit function key translation tables.

Under UNIX, the line terminator is a single \n, or LF character. Under AMOS and Windows, a CR-LF character pair is used. VUE is the only A-Shell module which under UNIX will automatically convert from the either format and will always save files in the current native format.

Special Commands

A-Shell/Vue is mainly intended for the convenience of those already familiar with some other version of VUE, (e.g. AlphaVue, XED, or d/VUE). Since the A-Shell/Vue supports mainly just the "core subset" of features that most people use, there did not seem to be much need to document it in detail here. However, there are a few editing features that are either non-standard extensions or just plain exotic, such that many people are not aware of them. Since it would be a shame to live without these features, they are hereby revealed.

Push/Pop: Although these are supported in virtually all other versions of VUE, A-Shell/Vue offers a useful enhancement on the idea by providing a stack of up to 10 push/pop locations. The standard PUSH and POP commands access this stack in the usual LIFO fashion, but if you specify a numeric stack location (e.g. >PUSH 5) it uses that entry directly. Thus is becomes something more like a "bookmark". Note that whenever you directly access the push/pop stack this way, it resets the "stack pointer" to that location, which will affect the operation of any subsequent "unadorned" PUSH or POP command.

Edit History commands: In addition to the automatic edit history feature (see **UPDATE** and **AUTODELETE**) above, there are also a couple of features which ease the maintenance of manual edit history notation. One is based on XED, and is invoked with ^_E (Control-Shift-Underline followed by **E**), while the other is based on d/VUE and is invoked with ^SE (Control-S followed by **E).** They both start by searching the file for the current edit number as defined by either a VEDIT = ### statement or a PROGRAM xxxxx, #.##(###) statement. (VEDIT is the symbol used by AMOS assembly language source files to define the edit number. If you want to use it with other program types, you will probably need to comment out the line to avoid compiler troubles.) Both commands search first for VEDIT, then for PROGRAM. If either is found with an acceptable syntax, it reads the 3 digit edit number. Then, in the XED version, it increments the number within the VEDIT or PROGRAM statement and uses it to construct a 2-line edit history entry, which inserts at the line where the cursor was, using the following format:

```
![124] October 02, 1998 05:59 PM        Edited by jack
!    _
```

(The idea is that you will then add in your own information describing the purpose of the edit.)

In the d/VUE version, the current edit number is not incremented, and the only thing inserted into the file at the position of the cursor is the current edit number in [###] notation. This form is particularly useful for commenting individual changes in a program and associating them with the program's edit number.

Use the Command Mode HELP command to get a complete list of all the available commands.

Update Notes: Build 879, 12 Apr 04

The following were added:

- Implement primitive 'bookmarks': these are similar to the multiple push/pop capability except that there is no concept of a hierarchy or stack. ^SM creates a bookmark for the current line. ^SN jumps to the next bookmark. ^sC clears the bookmark from the current line. The feature is handy when working on several sections in a large file and you want to jump between them.
- Implement the append macro feature (^_A).
- ^_F formats the current paragraph. (Same as >FORMAT).

- (UNIX, standalone mode) A new ini.vue feature, BAKDIR=<nativedirectory>, causes the .BAK file to be put in the specified directory, rather than in current directory. This can help avoid confusion in cases where .BAK files might not be recognized properly by other people or programs.
- (standalone) The display of the file path on the status line is now more intelligent, expanding it out with the current dir info if not specified on the command line, and trimming the first part of it if too long. This is particularly helpful when simultaneously editing multiple copies of a file in different directories.

WSET

TRACKER is a Debug/MicroSabio product for AMOS which keeps track of all output sent to the screen. In doing so, it manages to implement the -1,148 (save screen area) and -1,149 (restore screen area) TCRT calls for those terminal types that do not perform these functions in firmware. Tracking is enabled and disabled with the **WSET** command.

A-Shell includes a functional equivalent of TRACKER that is permanently enabled, and so the -1,148 and -1,149 TCRT codes are always available for terminals on an A-Shell system. The **WSET** command is included mainly for compatibility and to prevent A-Shell rejecting it as an illegal command, though some of the operational switches are implemented. It always responds as the AMOS version does when tracking is already enabled:

```
%TRACKER flag already set
Allocated TRACKER buffer located - TRACKER 2.0C already installed
```

The TRACKER /?, /M, and /N operational switches have been implemented.

ZXPWIN

ZXPWIN {<pcdir=>} {(ext)} <file>

(UNIX only) **ZXPWIN** is a handy utility for A-Shell/UNIX users running the ZTERM terminal emulator. It transfers the specified file to the PC and launches it using the Windows application appropriate for the file type (thus combining two or three steps which might otherwise be done manually.)

If the optional <*pcdir*=> argument is not specified, then the file is transferred to the default directory, whatever that may be. Otherwise, it is sent to the specified PC directory.

> If you have ZTERM Build 143 or higher, you can use PC environment variables, such as TEMP, but enclosing them in % (e.g. ZXPWIN %TEMP%=ABC.TXT)

The optional *(ext)* argument may be used to force the source file extension to be changed during the transfer. This is particularly useful since the extension determines which Windows application will be launched. For example, if you wanted to transfer a BAS file to the PC and view it in Notepad, you would want to specify (TXT). Otherwise, the default application for BAS files would probably be the Visual Basic editor, which is probably not present, and even if so, is probably not what you wanted.

> Since parentheses are significant to most UNIX shells, if you are going to specify a ZXPWIN command as part of an ashell command line (which will be processed by the shell), you will need to enclose the parentheses in quotes (e.g. ashell ZXPWIN %TEMP%="(TXT)"ABC.BAS)

The *file* argument can be in native or AMOS format.

Examples

```
ZXPWIN TEST.DOC
```

Transfer the file TEST.DOC to the PC in the default transfer directory and launch the Windows application which is associated with DOC files (probably Word or WordPad).

```
ZXPWIN %TEMP%=(CSV)TEST.PRT
```

Similar to the previous example, except that we will send the file to the PC directory defined by the PC's TEMP environment variable, and we will rename the file TEST.PRT to TEST.CSV on the PC (so that we launch the spreadsheet program). This would be most appropriate if the TEST.PRT file was actually in comma-delimited format.

```
HOST ashell ZXPWIN %TEMP%="(DOC)"TEST.BAS
```

The above command launches a new instance of ashell, which in turn executes the same **ZXPWIN** operation as in the previous example. Note that we must enclose the (DOC) argument in quotes since the shell (which will process the ashell command line) would otherwise strip off the parentheses.

```
ZXPWIN C:\TEMP=(TXT)/vm/miame/miame.ini
```

This command transfers the miame.ini file (note the use of the native UNIX filespec) to the C:\TEMP directory on the PC, renaming it to miame.txt, and then launching the TXT file viewer (which is probably Notepad).

CMD and DO Files

A-Shell supports CMD and DO files, more or less just like AMOS does, with a few extensions which are noted in the following topics entitled Global DO File Parameters and Special $ Variables. For readers who may not be completely familiar with CMD and DO files, a brief overview is presented here.

CMD and DO files are basically equivalent to BAT files under DOS or shell scripts under UNIX. They are simply text files—with .CMD or .DO extensions—containing LIT commands which are executed as if they were entered at the command prompt. The only difference between CMD and DO files is that CMD files do not support argument substitution and macro expansion, whereas DO files, like their DOS and UNIX counterparts, do support these function. Note that DO files do not *have* to take advantage of argument substitution, thus making CMD files somewhat redundant.

In addition to the normal commands that make sense at the command prompt, CMD and DO files also support some special commands and symbols which only make sense within CMD and DO files, and which allow them to be somewhat more program-like. Consider the following sample DO file (MAIN.DO), then see the following sections for more information.

Example: MAIN.DO

```
$D 100 MENU                 ; defaults for $0 and $1
:T                          ; trace (echo) commands
LOG DSK0:[150,$0]
:R                          ; don't trace commands
IF LOOKUP ("MENU.DAT") = -1
:<Sorry, no MENU.DAT here
>
GOTO DONE
ENDIF
RUN $1
;DONE                       ; (this is a label)
IF "$NU" <> "admin"         ; force all users except "admin"
  HOST                      ; to exit A-Shell
ELSE
:<
Now returning to cm prompt...
>
ENDIF
```

Arguments $0 through $9

The formal arguments $0 through $9 within the DO file are replaced by the first through tenth space-delimited tokens on the command line that invokes the DO file, with the first token being the one following the name of the DO file itself on the command line. Thus, if the command line was:

```
DO MAIN 22 GL
```

then $0 will be replaced by "22" and $1 by "GL". This differs from DOS (which uses %0 through %9) and UNIX (which uses $0 through $9), both of which use the zero parameter to refer to the name of the batch or script file itself rather than the first user argument.

Since the space is used as a token delimiter, a special trick is needed to create a single argument that contains a space. Angle brackets may be used for this purpose to combine multiple tokens into a single argument. For example, in the following command line:

```
DO MAIN 22 <GL OPTION1 OPTION2>
```

"GL OPTION1 OPTION2" will be treated as $1.

To establish default values for arguments which are not specified on the command line, the DO file can begin with the special command $D followed by the defaults for $0, $1, etc. Arguments not specified in this list will have no default. You can also specifically skip over a default argument by inserting just a $ in the argument's position on the $D line. For example:

```
$D DEF0 $ DEF2
```

This would set the default for $0 to "DEF0", the default for $1 to nothing, and the default for $2 to "DEF2".

The $D command, if present, must be on the first line of the DO file.

Comments

The comment character is the semicolon (;), unlike UNIX which uses the pound sign (#) and DOS which uses "REM". The comment character may appear anywhere in the command line, with the exception that labels, which have the same syntax as a one-word comment, must start in column one. (See the line ";DONE" in the sample file **Error! Reference source not found.** above.)

Colon (:) Commands

Certain special commands begin with the colon, and these must start in the first column.

Command	Meaning
:< .... >	outputs the text between the opening and closing angle brackets; may span multiple lines. This is the normal way to print operational messages on the screen from within CMD and DO files.
:t (trace)	causes all commands from that point forward to be traced—i.e., be displayed on the screen. This is very useful for debugging, but may be too verbose for normal use.
:r (respond)	causes just the output of commands to be displayed; the commands themselves do not display. This produces a somewhat cleaner looking result.
:s (silence)	silences all output. Unfortunately, this is the default, if neither *:t* or *:r* are supplied first.
:x (exit)	causes the CMD or DO file to exit. This is equivalent to the EXIT command.

:k (keyboard)	stops and waits for keyboard input. The input is then processed by whatever process is currently in effect (i.e. either a program or the command line processor itself). For example:
	`SET CTRLC`
	`:<This command erases all files!!!`
	`Hit ENTER to proceed or ^C to abort: >`
	`:K ; ^C will abort here; ENTER will just proceed`
	`ERASE *.*`

:p (partial)	may be used in conjunction with :k to append the keyboard input to a partial command line. For example:
	`:R`
	`:<Enter name of file to display: >`
	`:P`
	`EZTYP`
	`:K`

IF / ELSE / ENDIF Statements

These statements are similar to their BASIC Plus counterparts. The following functions and expressions are supported by IF, with the same syntax as for BASIC:

ABS	ACS	ASC	ASN	ATN	CHR
COS	DATN	DEVICE	EXP	FACT	FIX
FIX	INSTR	INT	LCS	LEFT	LEN
LEN	LOG	LOG10	LOOKUP	MID	RIGHT
RND	SGN	SIN	SPACE	SQR	STR
TAN	TIME	UCS	VAL		

The most useful of the above functions, LOOKUP, works slightly differently than the BASIC Plus equivalent. Instead of returning the file size, it always returns –1 if the file exists, else 0. See the example in the **Error! Reference source not found.** file above.

The following relational operators are supported:

Symbol	Meaning
=	Equal
<>	Not equal
#	Not equal
>	Greater than
>=	Greater than or equal
<	Less than
<=	Less than or equal

The following mathematical operators are supported:

Symbol	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
^	Raise to power
**	Raise to power

GOTO and EXIT Statements

GOTO is similar to the BASIC GOTO statement, except that it can only jump forward in the CMD or DO file. The syntax of a label consists of a semicolon followed by the label name. (See the label ";DONE" in the sample **Error! Reference source not found.** file above.)

EXIT causes an immediate exit to the A-Shell command prompt, even if you are currently nested multiple levels deep in a CMD or DO file.

LOOKUP Statement

The **LOOKUP** statement is a variation of the LOOKUP function (see IF statements above). The syntax is:

```
LOOKUP <fspec>/Optional message if not found
```

If the specified file is not found, the optional message (following the /, which is mandatory) is displayed and execution continues with the following line. Otherwise, the following line is skipped. Typically this is used with GOTO to create a conditional branch, as in the following:

```
LOOKUP $0/$0 does not exist!
GOTO DONE                          ; (EXIT might also be appropriate here)
VUE $0
;DONE
```

Special $ Variables

The DO file processor supports several special information variables (or built-in macros) which take the form $x or $xx (where x is alphabetic). The supported variables are listed here; those that are A-Shell extensions are noted at the end of the list.

Variable	Value
$:	Original device (e.g. “DSK0:”)
$P	Original ppn (e.g. “7,6”)
$$	Real dollar sign
$TM	Time of day in 24 hour HH:MM format (e.g. “13:27”)
$TD	Current date in YYMMDD format
$TW	Current day of week as a number (0=Mon, 1=Tue,...)
$NJ	Current job name (e.g. “TSKAAA”)
$NT	Current terminal name
$ND	Current terminal driver name
$NS	Current OS (e.g. “AIX”)
$SV	Current OS version
$UX	Current radix (e.g. “8” or “16”)
$LG	Current language (e.g. “ENGLISH”)
$LY	Yes character (e.g. “Y”)
$LN	No character (e.g. “N”)
$NU	Current login name (e.g. “root”)

The following are supported only under A-Shell:

Variable	Value
$HF	Current host full path (e.g. “/vm/miame/dsk0/001004”)
$HD	Current host disk (e.g. “dsk0”)
$HP	Current host ppn (e.g. “001004”)
$HM	MIAME env var definition (e.g. “/vm/miame”)

These special information variables are typically used in IF statements or in output messages.

It is necessary to quote these variables when comparing then in IF statements. For example:

```
IF "$NU" = "jack"      ; (correct)
IF $NU = jack          ; (incorrect)
```

Global DO File Parameters

This is another A-Shell extension that functions something like the special $ information variables, except that their values may be set from within a BASIC program. Once set, they retain their value for the duration of that session (unless specifically over-written by a BASIC program). There are 10 possible global DO file parameters, and they may be invoked on a command line using %0 through %9. The DO file processor substitutes these formal parameters for their values before passing them on to the DO file. For example, consider the command line:

```
XYZ ONE %3 TWO %0
```

Assuming that global parameter %3 had previously been set to "gee whiz" and global parameter %0 had previously be set to "humdinger", then this command would be processed as if it were:

```
XYZ ONE <gee whiz> TWO humdinger
```

"ONE" would be substituted for $0, "gee whiz" for $1, "TWO" for $2, and "humdinger" for $3.

There is no particular limit on the size of these variables, although there is limit of 200 characters on the length of any expanded command line.

To set global parameters, use XCALL MIAMEX as follows:

```
xcall MIAMEX, 79, status, g0, g1, g2, g3, g4, g5, g6, g7, g8, g9
```

The values of the variables *g0* through *g9* (all of them optional, all of them string or unformatted) will be plugged into the corresponding global parameters. Any of the parameters *g0* through *g9* that are not specified will cause the corresponding variable to be cleared. *Status* (F,6) will be returned as 0 for success, else error.

See the sample program MIAMEX.BAS in [7,376] for an example of setting and even retrieving these global DO parameters. (Although they were designed to be used with DO files, there is nothing stopping you from using them for holding and passing parameters amongst your own programs.)

K0 - K9 Commands

This capability was added to A-Shell in build 909.7 of 19 December 2004.

The CMD/DO file processor now supports a variation of the :K command to input keyboard responses into variables which can tested in IF statements and used as DO file parameters. The following sample CMD file provides an example:

```
:R
:<Enter desired ppn: >
:K0

:S
:<Enter password: >
:K9
IF "$KBD9" = "SECRET"
DOLOG $KBD0
```

The special command :K0 is like :K, in that it waits for a line of input, but instead of passing the input to the current program, it simply stores it as $KBD0. The same concept applies to :K1 thru :K2 ($KBD1 thru $KBD9). The special $KBDn variables may be tested in an IF statement or passed as a command line argument to a DO file.

Note that when using $KBDn in an IF statement, it must be quoted (as in the example above.) However, when using it as a DO file command argument, it should not be quoted. (In the example above, the DO file DOLOG.DO will see the $KBD0 on its command line and convert it to whatever had been entered to the "Enter desired ppn" prompt.)

$KBDn variables cannot be used in any other context, even within a CMD/DO file. Unlike other special $xx variables which can appear in DO files, which are expanded by DO.LIT when it loads the DO file, the $KBD variables cannot be expanded until after the input operation takes place, and thus it is left to individual commands (currently just IF and DO) to interpret them.

This feature requires LIT files of at least the versions shown below:

```
IF.LIT 1.0(106)
DO.LIT 2.1(121)
MDO.LIT 2.1(121)
```

Input Case

This capability was added to A-Shell in build 909.7 of 19 December 2004.

DO.LIT 2.1(121) supports the ability to specify the upper or lower case version of a $KBDn variable on the command line, which can be handy when building case-sensitive UNIX commands from input strings. The following example makes this clear:

```
;TEST.CMD
:<Input something >
:K1
DOTEST $KBD1 LCS($KBD1) UCS($KBD1)

;DOTEST.DO
:<Command line: $0 $1 $2
>
```

Assuming that the user had input “SiLly” in response to the prompt in TEST.CMD, the output of the DOTEST command would be:

```
Command line: SiLly silly SILLY
```

Note that these two functions , LCS($KBDn) and UCS($KBDn), are the only ones supported on the DO command line.

Concatenate Parameters

This capability was added to A-Shell in build 910 of 3 January 2005.

DO.LIT 2.1(122) supports a mechanism for concatenating parameters on the command line (before they are bound to the $0 thru $9 formal parameters). The syntax consists of prepending $+ to the start of a parameter that you want to combine with the prior parameter. For example, consider a generic DOTHIS.DO file consisting of the following command:

```
HOST $0 $1 $2 $3 $4 $5 $6 $7 $8 $9
```

If you executed it with:

```
.DOTHIS play ina $+gada $+davida real loud
```

The resulting host command would be:

```
play inagadadavida real loud
```

Note that “real” and “loud”, which started out in positions $4 and $5, got shifted by the concatenation down to $2 and $3.

You can also use the $+ operator in the middle of an argument to avoid the argument renumbering problem just described, i.e.:

```
.DOTHIS play ina$+gada$+davida real loud
```

In this case, it is more clear that there are only 4 arguments ($0 thru $3) being passed to DOTHIS. This technique is mainly of interest in conjunction with $KBDn variables and the UCS/LCS functions where you may want to concatenate some folded keyboard input with a hardcode value, and doing do directly would interfere with the interpretation of the $KBDn and LCS() / UCS() syntax. For example:

```
:R
:<Enter command: >
:K0
:<Enter ini file: >
:K1
DOTHIS /vm/miame/bin/ $+LCS($KBD0) -i /vm/miame/ $+LCS($KBD1)
```

Assuming the user input “ASHELL” to the first prompt and “MIAME.INI” to the second, then the DOTHIS.DO file would execute:

```
HOST /vm/miame/bin/ashell -i /vm/miame/miame.ini
```

Without the concatenation, the user would have had to enter the entire paths or do with the LCS() function.

Index

A

ABOUT command, 17
ABSLOOKUP, 39
ABSLOOKUP (SET option), 73
Access methods, 39
AHASH (SET option), 73
Allocate, 39
Allocated jobs, 83
AlphaBASIC extensions, 31, 37
AlphaBASIC Plus, 25
AMOS (TRACE option), 77
AMOS statement, 28
AMOSRUNSBR (SET option), 73
AMSORT (TRACE option), 77
Angle brackets, 101, 102
APPEND command, 18
ASCII numeric values, 29
ASHCFG account defined, 8
A-Shell Editor, 46
A-Shell terminal emulator, 20
Ashkill, 61
ASHLOG.LOG, 84
ASTAT command, 19
ATE, 20, 78, 85
ATE (TRACE option), 77
ATECFG command, 20
ATECTL command, 20
AUTODELETE (VUE command), 93
AUTOPRINT (SET option), 73

B

Background jobs, 80, 83
BAS account defined, 9
BASERR (TRACE option), 77
BASIC extensions, 31, 37
BASIC Plus, 25
BAT files, 100
Bell, 41
Binary compatibility, 11
Blocks, 39
Bookmarks (VUE), 96

C

Cache, 65
Case, 108
Case sensitivity, 8, 10
cat (UNIX command, 87
CDATE switch on DIR.LIT, 43
CHAIN, 26
CHARINS (VUE command), 92
CHAT command, 21
CLS, 14
CMD account defined, 8
CMD files, 100
CMDINP (SET option), 73
Colon commands, 102
Color, 75
COLUMN (VUE command), 92
Columns, 51, 75
Comma delimited files, 35
Command file input, 74
Command files, 100
Command syntax, 10
Compare files, 79, 88
Compatibility, 11
COMPIL command, 22
COMPIL switches, 23
COMPLP, 14
Concatenate file, 18
Concatenate parameters, 109
Conditionals in CMD/DO files, 47, 58
Console name, 19
Constants, 29
Contiguous files, 39

Control characters, displaying, 87
Control files, 80
Control keys for editing, 13
Control-C, 61
COPY command, 38
CREATE command, 39
CR-LF character pair, 95
cron, 82
CSV files, 33
CTRLC (SET option), 73
Cursor control, 13

D

d/VUE, 93, 96
DATE command, 39
DEBUG, 32
DEBUG (SET option), 73
DEBUG (TRACE option), 77
DEFINE, 29
DEL command, 40
Device mapping, 9
Device name (terminal), 84
Devices, 8, 40
Devices, listing, 83
DEVTBL command, 40
DIFF command, 41
Difference in files, 79
DING command, 41
DIR command, 42
DIR/V, 90
DIRECT.LST, 38
Directory listing, 88
Disk block dump, 45
Display file, 87
Display height, 51
Divide by zero, 29
DO command, 44
DO file parameters, 106
DO files, 100
DO loop, 27
Dot variable, 37
DPRINT, 32
DRAW (VUE parameter), 94
DUMP command, 45
Dump ISAM file, 60
Dump to screen, 68
DVR account defined, 8
Dynamic array allocation, 32

E

ECHO (SET option), 73
Edit history, 96
EDIT$ function, 26, 27
Editing, command line, 13
Editor, 46
EL command, 46
ELSE command, 47
ELSE statements, 103
End of file marker, 18
ENDIF, 26
ENDIF statements, 103
EP command, 46
ERASE command, 48
ERSATZ command, 48
ERSATZ devices, 8
EXIT command, 48
EXIT statement, 26
EXIT statement file, 104
Extensions, 8
Extensions to AlphaBASIC, 25, 31, 37
EXTFIO (SET option), 73
EZPRT, 15
EZTYP command, 49
EZVUE, 49

F

FIELDEMU (SET option), 73
File
  access methods, 39
  dump, 45
  locking, 64
  specs, 8, 12
  types, 39
  version, 90
FILL$ function, 26
FIXTRN command, 52
FLIT, 15
FOPENS (TRACE option), 77
For...next loop, 27
FORCE command, 54
Foreground jobs, 83
FPROUND (SET option), 73
FREE, 15
Function key translations, 8, 28, 52, 95

G

GDIPRT (TRACE option), 77
GETKEY statement, 28

GO (VUE parameter), 94
GOTO, 15
GUI (TRACE option), 77

H

Hard page breaks, 49
Hash code, 73, 88
HEAD, 15
HEAD command, 56
Height, terminal, 75
HEX (SET option), 73
Hex values, 29
History, 96
HOST command, 57
Host commands, 57

I

IEDIT, 27
IF command, 58
IF statements, 103
IF...ENDIF, 26
IFX, 52
IGNORE.PRG, 19
INCLUDE nesting, 36
Include++, 22, 68
INFDEF (SET option), 73
Informix, 60
INI.VUE, 92
Input CSV, 33
Input raw, 26
INSERT (VUE command), 92
Integer variables, 28
Inter-job communications, 60, 61
ISAM (TRACE option), 77
ISAM, ISAMPLUS, 22, 24, 26, 59
ISMBLD, 15
ISMUTL command, 59

J

Job information, 19
JOBALC, 15
JSTAT command, 60

K

Key, 62
Keyboard (VUE command), 102
KILL command, 61
Kill job, 61
Kill task, 80

L

Landscape, 69
LATIN1 (SET option), 73
Launch process, 80
LF characters, 95
LIB account defined, 9
LICENS command, 62
License info, 17
License update, 62
Line terminator, 95
List tasks, 80
LOAD command, 63
Locking, 28, 60, 64
LOCKS (TRACE option), 77
LOG command, 64
Logical units, 8
Login account, 64
Login name, 21, 105
LOGOFF, 15
LOKSER, 64, 74
LOKSER (SET option), 73
LOKUTL command, 64
LONG switch on DIR.LIT, 43
LONGDIR (SET option), 73
LOOK, 49
LOOKUP, 15
LOOKUP statement, 104
Loop statements, 27
LP (TRACE option), 77
LSTLIN.SBR, 29

M

Macro, 29
MAKE, 15
MALLOC (TRACE option), 77
MAP command, 65
MAPIDA, 60
Master directory listing, 89
Maximum modules, 63
MDO, 15
MEMORY command, 65
Messages, 102
MFD, 45
MMAP command, 66
MODEM (VUE parameter), 94
MOVE command, 67

N

NARROW (VUE command), 92

O

OCMPIL, 15
OCTAL (SET option), 73
Octal values, 29
ODTIM, 27
OPR account defined, 8
OPRE, 15
OPRE command, 68
Orientation, 69
ORUN, 15
Outfile, 10

P

PAD, 33
Page breaks, 49
PAGE command, 68
Page mode, 87
Parameter passing (to SBX), 33
Partition, 8, 65
Password, 8, 16
PFK modules, 52, 63
Phantom jobs, 84
Platforms, 11
PolyShell, 72
POP (VUE command), 96
PPN, 15
PPNs, 8
PRAGMA, 29
PRE, 15
PRE command, 68
PREBAS, 68
PRINT command, 69
Print orientation, 69
Print to screen, 87
Printing, EZTYP, 50
Privileges, 8, 21, 43, 54, 60, 61, 72
Program name, 19
Program version, 90
PROMPT (SET option), 73
PUSH (VUE command), 96
PWD, 15

Q

Qflock.sys, 70
QUIT, 15
QUITWARN (VUE command), 94
QUTL command, 70

R

RAD50, 8
Radix, 29
Random files, 39
READ'ONLY, 28
Recall command line, 13
Remote client, 86
RENAME command, 71
Repeat a command, 87
Resource info, 90
Respond (VUE command), 102
RETURN(expression), 33
RP file format, 26
RW (TRACE option), 77

S

SAVE command, 71
Screen dump, 68
Screen messages, 102
Scrolling, 68
Search path, 11
SEARCHFOLD (VUE command), 92
Security key, 62
SEND command, 72
Sequential files, 39
SET command, 73
Shell scripts, 100
Shifted status line, 94
SHORT (VUE command), 92
SIGHUP (TRACE option), 77
SIGNAL (TRACE option), 77
Silence (VUE command, 102
SIZE, 16
SLEEP, 16
Sleep interval, 19
SLEEPR, 16
Slow mode, 87
SOCKS (TRACE option), 77
Software license, 62
SORT, 16
Sound, 41
Source file compare, 79
SRCCOM command, 41
START (VUE command), 92
Status line, 49, 94
Status, job, 60

Status, system, 83
STRICT (SET option), 73
Strip, 26
SUBMIT command, 80
SUBMIT.LIT, 80
Switches, defined, 10
Syntax, command, 10
SYS account defined, 8
SYSACT, 16
SYSERR (TRACE option), 77
SYSTAT command, 83
System memory, 52
System status, 83

T

TAB (VUE command), 92
Tab delimited output, 32
TAIL command, 85
TALL (VUE command), 92
Target job, 61
Target name, 21
Task manager, 80, 82
TELNET command, 85
TELNET configuration, 20
Telnet server, 86
TELSER command, 86
Terminal
  device i.d., 84
  drivers, 95
  emulator, 20
  handling, 57
  width and height, 75
TERMINAL (SET options), 73
Terminate ATE session, 85
Termination in CMD/DO files, 48
Termination, job, 61
TIMEcommand, 39
TITLE, 76
TITLE (SET option), 73
Token delimiter, 101
TPRINT, 32
TRACE (SET option), 73
Trace (VUE command), 102
TRACE options, 77
Tracker, 57, 98
tty name, 19
TYPE command, 87

U

U command, 87
UFD, 45
UPDATE (VUE command), 93
User
  ID, 21, 61, 72
  memory, 52, 63, 65, 71
  name, 19, 21
USRMEM (TRACE option), 77

V

Variable rows, 51
Variables in CMD/DO files, 105
VER, 16
VERIFY command, 88
Version info, 90
VERSYS command, 90
VUE command, 91
VUX files, 52, 95

W

WAV file, 41
WHILE Loop, 27
WIDE (VUE command), 92
Wide reports, 51
Width, terminal, 75
Wildcards, 12
WRAP (VUE command), 92
WRITECD, TD, 35
WSET command, 98

X

XCALL (TRACE option), 77
XED, 91
XFUNC, 36
XGETARG, 33
XPUTARG, 33
XY, 16

Z

Zap a user, 70
Zero parameter, 101
Zombies, 81, 84
ZTERM, 20, 98
ZXPWIN command, 98